

LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

AD-A239 000



DTIC ACCESSION NUMBER

LEVEL

INVENTORY

WL-TR-91-8025

DOCUMENT IDENTIFICATION

Jul 1991

DISTRIBUTION STATEMENT

ACCESSION FOR

NTIS ☐ GRA&I ☒
DTIC ☐ TRAC ☐
UNANNOUNCED ☐
JUSTIFICATION ☐

BY

DISTRIBUTION/

AVAILABILITY CODES

DISTRIBUTION

AVAILABILITY AND/OR SPECIAL

A-1

DISTRIBUTION STAMP

91 7 0 067

91-06506



DATE RECEIVED IN DTIC

DATE ACCESSIONED

DATE RETURNED

REGISTERED OR CERTIFIED NUMBER

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

H
A
N
D
L
E

W
I
T
H

C
A
R
E

AD-A239 000



PRODUCT DEFINITION DATA INTERFACE (PDDI)

Access Software User's Manual

McDonnell Aircraft Company
McDonnell Douglas Corporation
P. O. Box 516
St. Louis, MO 63166

July 1991

Final report

Approved for public release; distribution is unlimited.

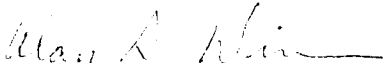
MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.




ALAN R. WINN
Project Manager

30 May '91

DATE

FOR THE COMMANDER:



BRUCE A. RASMUSSEN, Chief
Integration Technology Division
Manufacturing Technology Directorate

31 May 91

DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/MTIB, WPAFB, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S) WL-TR-91-8025		
6a. NAME OF PERFORMING ORGANIZATION McDonnell Aircraft Company		6b. OFFICE SYMBOL (If applicable) McAir	7a. NAME OF MONITORING ORGANIZATION Manufacturing Technology Dir. (WL/MTIB) Wright Laboratory		
6c. ADDRESS (City, State, and ZIP Code) McDonnell Douglas Corporation P. O. Box 516, St. Louis, MO 63166			7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, OH 45433-6533		
8a. NAME OF FUNDING/SPONSORING		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-82-C-5036		
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson Air Force Base, Ohio 45433-6533			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO. 78011F	PROJECT NO. 3095	TASK NO. 06
11. TITLE (Include Security Classification) PRODUCT DEFINITION DATA INTERFACE (PDDI), Access Software User's Manual					
12. PERSONAL AUTHOR(S) (see reverse side)					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Year, Month, Day) July 1991	15. PAGE COUNT 144
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Produce Definition Data ICAM Architecture		
			Life Cycle Document CAD/CAM		
			Engrg./Mfg. Interface (continued on back)		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This document is the Access Software User's Manual for the Product Definition Data Interface (PDDI) Extensions contract. This document provides procedures for Application Programmers to use the PDDI Access Software. User's Manual UM560130000A provides procedures for use of the PDDI Translator. ←					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> OTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Alan Winn			22b. TELEPHONE (Include Area Code) (513) 255-8787	22c. OFFICE SYMBOL WL/MTIB	

12. Personal Author(s):

Chi, Kelly
Baldrige, Gary
Magnuson, Charles
Mehl, Kenneth
Oakes, Janet
Shreve, Edward
Ulmer, Beth
White, George

18. Subject Terms:

Needs Analysis Document
System Requirement Document
State-of-the-Art Document
System Specification Document
SS - Draft Standard
System Design Specification
Product Specification
Operators Manual
Users Manual - Translator

UM 560130001
1 January 1987

FOREWORD

This document was produced under Air Force Contract F33615-82-C-5036, Product Definition Data Interface (PDDI). This contract is sponsored by the Air Force Wright Aeronautical Laboratories, Materials Laboratory, Air Force Systems Command, Wright-Patterson, Air Force Base, Ohio.

This program is being administered under the technical direction of Lt. Eric Gunther, ICAM Project Manager. The MCAIR Program Manager is Mr. Jerry Weiss and Mr. Herb Ryan is the Deputy Program Manager.

This document was prepared in accordance with the ICAM Configuration Management Life Cycle Documentation requirements for the Configuration Item.

TABLE OF CONTENTS

	<u>Page</u>
FOREWORD	i
1.0 SCOPE	1-1
1.1 IDENTIFICATION.	1-1
1.2 INTRODUCTION.	1-1
1.3 OTHER SYSTEM MANUALS.	1-1
1.4 APPROACH.	1-1
2.0 REFERENCES	2-1
2.1 APPLICABLE DOCUMENTS.	2-1
2.1.1 SPECIFICATION	2-1
2.1.2 STANDARDS	2-1
2.1.3 OTHER PUBLICATIONS.	2-1
2.2 TERMS AND ABBREVIATIONS	2-3
3.0 SYSTEM OVERVIEW	3-1
3.1 INTRODUCTION.	3-1
3.2 SYSTEM INTERFACES	3-1
3.3 SYSTEM ENVIRONMENT.	3-1
3.4 PHYSICAL SCHEMAS.	3-2
3.5 SOFTWARE PACKAGES	3-2
3.6 TRANSLATOR.	3-5
3.7 MODEL ACCESS SOFTWARE	3-7
3.8 DATA ITEMS.	3-8
3.9 INTERFACE PARAMETERS.	3-11
3.10 MEMORY MANAGER.	3-13
4.0 INTERFACE OPERATIONS	4-1
4.1 INTRODUCTION.	4-3
4.2 INITIALIZATION/DELETION OF THE ACCESS SOFTWARE ENVIRONMENT	4-4
4.3 ENTITY OPERATIONS	4-7
4.3.1 CREATE OPERATIONS	4-8
4.3.2 GET OPERATIONS.	4-16
4.3.3 UPDATE OPERATIONS	4-19
4.3.4 DELETE OPERATIONS	4-21
4.3.5 ACTIVATE OPERATIONS	4-29
4.3.6 APPLICATION FLAG OPERATIONS	4-33
4.4 LIST OPERATIONS	4-42
4.4.1 BOOLEAN OPERATIONS.	4-43
4.4.2 STRUCTURE OPERATIONS.	4-47
4.4.3 GENERAL OPERATIONS.	4-50
4.4.4 DELETE OPERATIONS	4-57
4.4.5 EDIT OPERATIONS	4-62
4.4.6 SEQUENTIAL READ AND EXECUTE OPERATIONS.	4-72

TABLE OF CONTENTS

	<u>Page</u>
5.0 GENERAL UTILITIES	5-1
6.0 SAMPLE PROGRAMS	6-1
7.0 APPENDICES	
CALLING PARAMETER INDEX	A-1
ROUTINE INDEX	B-1
RETURN CODE INDEX	C-1
FORTRAN SCHEMA DIAGRAM.	D-1
PASCAL SCHEMA DIAGRAM	E-1
GENERAL TECHNIQUES/GUIDELINES	F-1
RUN-TIME ENVIRONMENT.	G-1
MAS ERROR AND WARNING RETURN CODE INDEX	H-1

USER'S MANUAL

SECTION 1

SCOPE

1.1 Identification

This User's Manual provides a guide for the use of ACCESS Software developed for the Product Definition Data Interface (PDDI) Project 5601. This project was developed under Air Force Contract F33516-82-C-5036.

1.2 Introduction

Capabilities documented in this manual include:

- o Access Software Initialization
- o Entity Operations
- o List Operations

This PDDI software operates on IBM 43xx, 308xx or DEC VAX 11/780 computers. The environmental requirements are described in Section 3.

The PDDI system documentation does not address local (native) system or computing environment documentation.

This manual addresses IBM procedures and terminology only.

1.3 Other system Manuals

The associated Operator's Manual (OM) describes the system operation and installation procedures. It is intended for use by computer operators and programming personnel.

An associated User's Manual (UM 560130000) is provided for users of the PDDI Translator.

The PDDI Product Specification (PS) provides routine descriptions, data dictionary listings and PDDI system messages for system maintenance purposes.

1.4 Approach

This User's Manual is divided into six (6) main sections:

- Section 1 - Scope of this document.
- Section 2 - Reference documentation applicable to PDDI and this document.
- Section 3 - The PDDI architecture at a high level and introduction to the use of the Access Software.
- Section 4 - Entity and List Operations needed to access the data structures passed back to the Application program.
- Section 5 - Descriptions of the general utilities routines available with the Access Software.
- Section 6 - Sample programs using the Access Software in Pascal and FORTRAN.
- Appendix A - Access Software Calling Parameter Index
- Appendix B - Alphabetical Access Software Routine Index
- Appendix C - Access Software Return Code Index
- Appendix D - Access Software FORTRAN Schema Diagram
- Appendix E - Access Software Pascal Schema Diagram
- Appendix F - General Techniques/Guidelines
- Appendix G - Run-Time Environment
- Appendix H - Error and Warning Return Code Index

SECTION 2

REFERENCES

2.1 Applicable Documents

2.1.1 Specification:

DOD-D-1000B
MIL-D-5840

Drawings, Engineering and Associated Lists
Requirements for Data, Engineering and Technical
Reproduction

2.1.2 Standards:

ANSI Y14.5
ANSI Y14.26M

Dimensioning and Tolerancing
Digital Representation
Communication of Production
Definition Data

ANSI B46.1

Surface Texture (Surface Roughness,
Waviness and Lay)

ANSI B92.1
DOD-STD-100C
MIL-STD-9

Involute Splines and Inspection
Engineering Drawing Practices
Screw Thread Conventions and Methods
of Specifying

MIL-STD-12

Abbreviations for Use on Drawings,
Specifications, Standards and in
Technical Documents

IDS150120000C
IEEE STD 829

ICAM Documentation Standards
Standards for Software Test
Documentation

ISO/TC184/SC4/WG1

4.2:2 The Step File Structure (Working Paper
Version 1.0 28 April 1981)

2.1.3 Other Publications:

CLD150120000
FTR110210000U
FTR110232000U

ICAM Document Catalog
ICAM Architecture
ICAM Architecture Part II, Automated
IDEFO Development

Product Definition Data Interface

ITR560130001U	First Interim Technical Report (Period 1 Oct 82 - 31 Dec 82)
ITR560130002U	Second Interim Technical Report (Period 1 Jan 83 - 31 Mar 83)
ITR560130003U	Third Interim Technical Report (Period 1 Apr 83 - 30 June 83)
ITR560130004U	Fourth Interim Technical Report (Period 1 Jul 83 - 30 Sep 83)
ITR560130005U	Fifth Interim Technical Report (Period 1 Oct 83 - 1 Dec 83)
ITR560230006U	Sixth Interim Technical Report (Period 1 Jan 84 - 31 Mar 84)
ITR560130007U	Seventh Interim Technical Report (Period 1 Apr 84 - 30 Jun 84)
ITR560130008U	Eighth Interim Technical Report (Period 1 Jul 84 - 30 Sep 84)
ITR560130009U	Ninth Interim Technical Report (Period 1 Oct 84 - 31 Dec 84)
ITR560130010U	Tenth Interim Technical Report (Period 1 Jan 84 - 31 Mar 85)
FTR560130001U	Task I, Final Technical Report - System Test Methodology, Volume III Technical Operating Report - Product Assurance/Quality Assurance - 15 Oct 85
SD 560130001U	Scoping Document
NAD560130000	Needs Analysis Document
SAD560130000	State-of-the-Art Document
SRD560130000	System Requirement Document

SDS560130000	System Design Specification Document
SS 560130100	System Specification Document
SS 560130200	System Specification Document - Draft Standard
STP560130000	System Test Plan
STR560130000	System Test Report
PS 560130000	Product Specification
OM 560130000	Operator's Manual
UM 560130000	User's Manual (Translator)

2.2 Terms and Abbreviations

The following list explains terminology, acronyms, and other abbreviations used in this document.

ACCESS SOFTWARE - A set of routines for creating, managing and querying an incore Working Form model.

ANSI - American National Standard Institute.

APPLICATION - Refers generically to any software modules which are used in CAD/CAM functions.

APPLICATION REQUEST - A request initiated by an application program, either through batch or interactive processing, which will interrogate the model through the PDDI Access Software to obtain or operate on specific information regarding the model and its components or elements.

APPLICATION REQUESTED DATA - The data which fulfills the application's original request and which is in the proper format and readable by the application.

ASCII - American Standard Code for Information Interchange.

ATTRIBUTE - An item of information about an entity. A key attribute identifies the entity; a role iterate gives a fact about an entity.

CAD/CAM - Computer Aided Design/Computer Aided Manufacturing.

CLASS - A collection of entities that are alike in some manner.

CLIST - IBM Command lists.

CONSTITUENT - A specific instance of an entity that is used in the definition of some other entity.

CONTEXT-FREE GRAMMAR - The syntax of the language gives a precise specification of the data without interpretation of it.

DOMAIN - The set of values permissible in a given context. A natural domain is the value set native to a given machine architecture; an imposed domain is a specific subset of the natural domain.

DYNAMIC ALLOCATION - The allocation (and deallocation) of memory resources as required by the application. The opposite is static allocation where a fixed size segment of memory is available to the application.

EBCDIC - Extended Binary Coded Decimal Interchange Code (IBM character set).

ENTITY - A collection of facts (attributes) about something of interest.

EXTERNAL REFERENCE - A reference to some quantity of data that exists somewhere outside the scope of the immediate body of information.

FUNCTIONALITY - (1) To show that the configuration item has fulfilled the specified requirements. (2) The receiving and sending systems can operate on the entity in the same manner with the same results within a pre-defined tolerance.

INCLUDE FILE - Pascal source code from another file or library included on the compilation of a Pascal source file.

INPUT DATA - That information which the application needs to supply in order to interrogate or operate on the model. This data may assume only these forms prescribed by the PDDI Access Software specifications.

INTERPRETED REQUEST - Input data which has been appropriately modified to conform to the PDDI Access Software's internal data representation so that it may be further processed.

JCL - Job Control Language - IBM language used to identify a job and describe its requirements to an operating system.

KEY - An item of data that uniquely identifies some specific instance of an entity.

MAS - MCAIR's acronym for the PDDI Access Software (Model Access System).

METAMODEL - A body of data that defines the characteristics of a data model or structure.

MODEL - A collection of PDD that is transferable, displayable, accessible, and equivalent to a Part. The internal representation of the application data, as initiated and organized by the user. The model is also referred to as the Working Form.

MODEL NETWORK DEFINITION - The set of rules and definitions which outline in detail the data structure whereby higher order entities may be composed of lower order entities, or constituents, and the lower order entities may be constituents of one or more higher order entities.

NATIVE SYSTEM - The PDD and applications in a format that is unique to the database of a CAD system.

PARSE - The process of analyzing input strings (records) to identify fields and to verify that the data has a valid format.

PDD - Product Definition Data.

POST-PROCESSOR - A phase of the translator where data is received from the Exchange Format and is converted to the Working Form.

PRE-PROCESSOR - A phase of the translator where data is taken from the Working Form and is converted to the Exchange Format.

QUALITY - The composite of all the attributes or characteristics including performance of an item or product.

QUALITY ASSURANCE - The planned and systematic establishment of all actions (management/engineering) necessary to provide adequate confidence and nonconformance prevention provisions and reviews are established during the design phase and performed throughout the software development and life cycle phases.

QUALITY CONTROL - The planned and systematic application of all actions (management/technical) necessary to control raw materials or products through the use of test, inspect, evaluate, and control of processes.

REQUESTED DATA - See Application Request Data.

RUN SYSTEM - The Translator sub-package which provides the communication interface between the user and the pre/post-processors.

SCHEMA - Those definitions which describe the content of the data and the relationship between the various elements or components of the data.

SOFTWARE QUALITY ASSURANCE (SQA) - The planned and systematic establishment of all actions necessary to provide adequate confidence that nonconformance prevention provisions and reviews are established during the design phase and performed throughout the software development and life cycle phases.

SOFTWARE QUALITY ASSURANCE PLAN (SOAP) - An organized description of the methods, policies, and procedures necessary to conduct software quality assurance and control activities during the design, development, delivery, and maintenance phases.

SOFTWARE QUALITY CONTROL - The planned and systematic application of all actions (management/technical) necessary to ensure that the software under development or maintenance satisfies the technical requirements through the use of tests, demonstrations, inspections, evaluations, and control of processes.

SYSTEM CONSTRAINTS - Those hardware and software environmental constraints which will be imposed upon the PDDI Access Software that will limit its implementation and application. An example of such constraints might be the particular compiler used to compile the PDDI Access Software package.

TRANSFER DATA - The data required to make an exchange of data between systems (e.g., delimiters, record counts, record length, entity counts, numeric precision).

TRANSLATOR - A software MECHANISM that is used for passing data between the Exchange Format and Working Form of the PDD.

TREE - A collection of the data that makes up an instance of an entity. The information is stored as records in a linked list.

TREE STRUCTURE - The arrangement of information within a tree.

TSO - Time Sharing Option - IBM function which provides conversational time sharing from remote terminals.

USER COMPUTER SYSTEM - The specific hardware, operating systems, and applications software systems that the user will employ to implement the PDDI Access Software.

WORKING FORM - A memory resident form of a model that supports rapid access to entities via the Access Software.

WORKING FORMAT - The physical representation of the Working Form within the computer.

SECTION 3

SYSTEM OVERVIEW

3.1 Introduction

The purpose of the PDDI Software System is to provide a prototype for the communication of complete Production Definition Data (PDD) between dissimilar CAD/CAM Systems. This system will serve as the information interface between Engineering and Manufacturing functions. It is composed of Access Software, Conceptual Schema, Exchange format and a Translator. (See Figure 3-1).

The Access Software is a set of callable utility programs that will allow applications to manipulate and query PDD. The Conceptual Schema contains the human readable data needed to define a CAD/CAM model. The Exchange Format is a neutral physical sequential format for passing data between dissimilar systems. The PDDI Translator is the software mechanism for passing this data between the Exchange Format and the Working Form of the PDD.

3.2 System Interfaces

The PDDI software must interface with the computer system on which it is installed, the local (native) CAD/CAM database, the Exchange Format, the Working Form, and the user (application). It does this via PDDI Access Software, the PDDI Translator and local (native) developed software packages. Note: Simple interim database software is included in the Translator software. This software is an interim program to be used until an interface to the native database system is available. See Appendix D for an explanation. Figure 3-3 shows the environment in which the PDDI system was developed. This figure also shows the versatility of the system and the multi-hardware environment in which it may be used. The left-hand side of Figure 3-3 shows the PDDI development environment.

3.3 System Environment

The PDDI system was developed in the following computing environment:

Computer/Operating System

IBM 43XX/MVS with TSO and associated tape drives, disk drives and terminals.

DEC VAX 11/780 VMS with associated tape drives, disk drives and terminals.

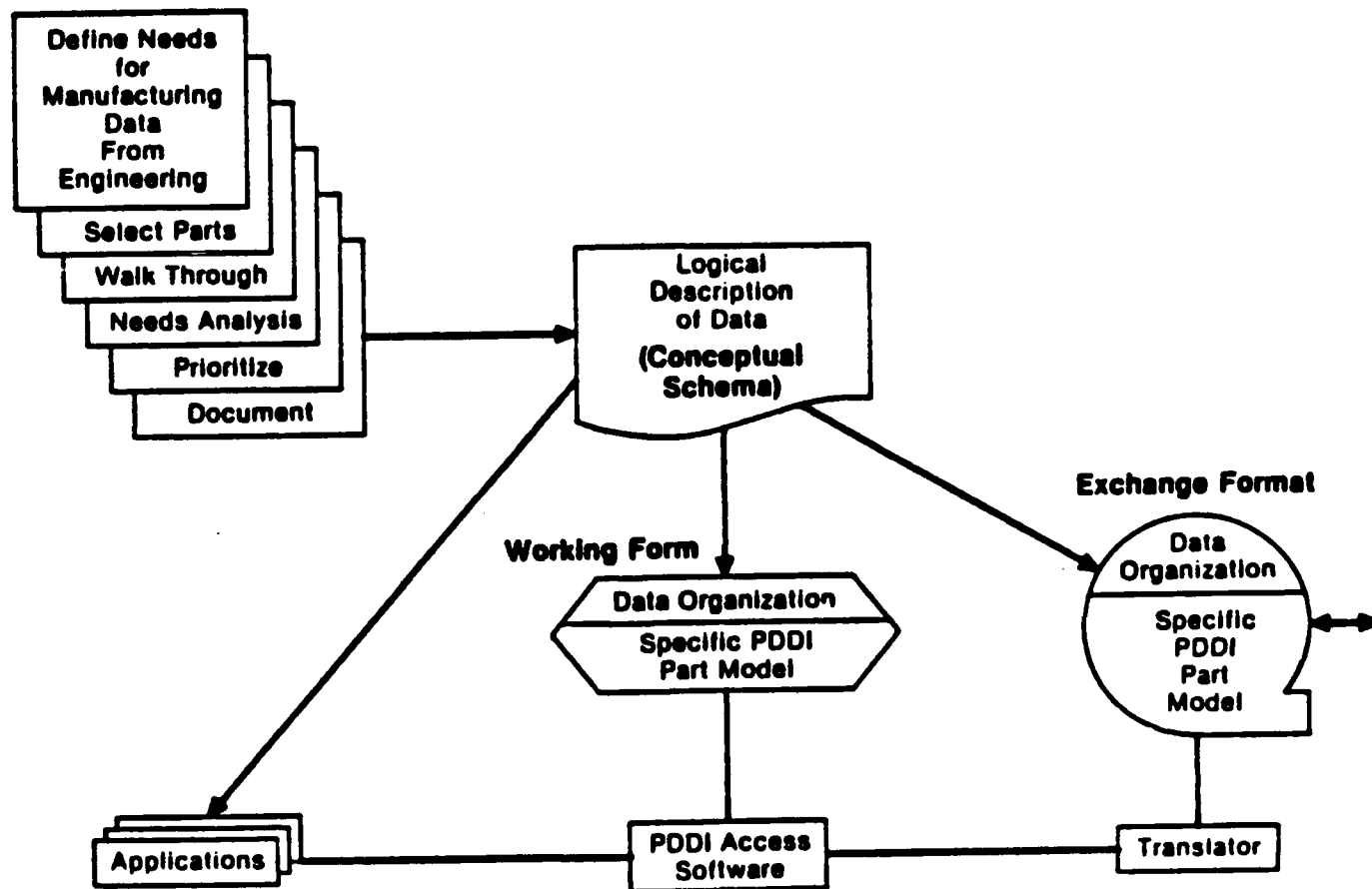


Figure 3-1 PDDI System Architecture

Storage (Core) Requirements

The minimum core requirements for the PDDI software and database is 1.0M plus the size of the model. (The PDDI Mechined Rib model required .57M)

* PDDI Machine Rib

Compilers

IBM-PASCAL/VS Release 2.2
DEC-PASCAL V3.3, FORTRAN 77 V4.4

Terminals

E&S PS300 (or equivalent for graphics applications)
IBM 3270 (or equivalent)

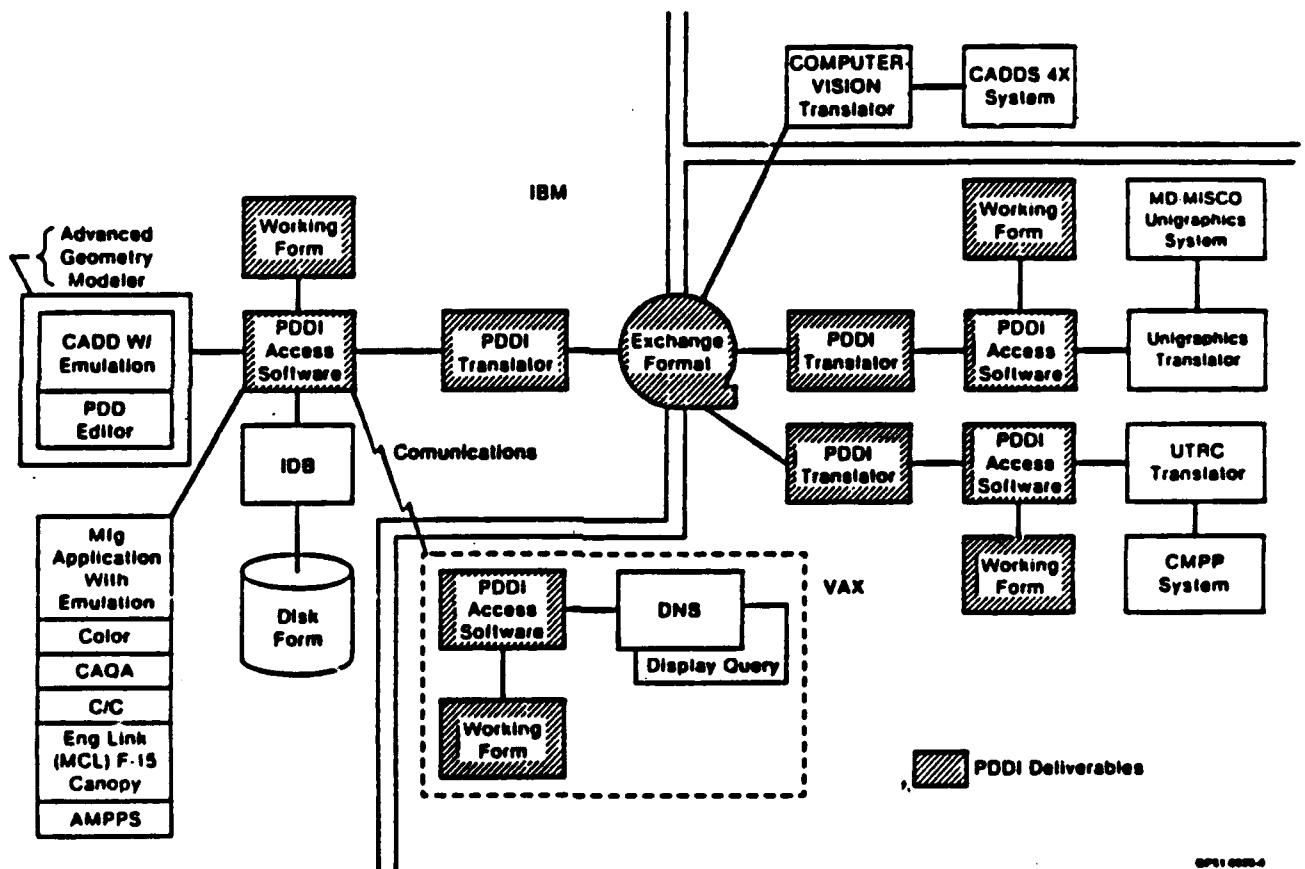
The PDDI system is transportable to other computing systems. However, appropriate local (native) interfaces (translator) must be provided. The right-hand side of Figure 3-2 shows the PDDI commercial demonstration architecture for UNIGRAPHICS and Computervision Systems.

3.4 Physical Schemas

The Working Form physical schema is determined through a data dictionary or PASCAL include files. An explanation of the form and the use of these files can be found in Appendix A. The Exchange Format physical schema is defined by the PDDI conceptual schema and the specification for the neutral file format.

3.5 Software Packages

The software for the system consists of two (2) packages - Access Software and Translator.



GP11 0000-4

Figure 3-2 PDDI Environment

3.6 Translator

The PDDI Translator is the software package used to format PDD for transmission between systems. The Translator is broken up into three main sub-packages. These sub-packages are: "Run System", "Pre-Processor" and "Post-Processor". (See Figure 3-3).

The Run System is the interface between the user and the "processors". This package provides menus, queries and system responses for the user.

Functions performed by this package include: Perform system configuration activities, determine files needed by the processors and make them available, and provide messages to aid user interfaces.

Access to the native database is also provided by this package via calls to user-supplied routines. Data from this database is placed into or obtained from the Working Form using calls to the Access Software. The pre-processor or post-processor is then called to perform the desired translation.

The Pre-Processor provides the interface from the Working Form to the Exchange Format.

Working Form entities, in the Working Form physical schema, are accessed via the Access Software. Tables, obtained from the Run System, are then used to map the Working Form entities to the Exchange Format physical schema. The Exchange Format entities are then encoded and placed into the Exchange Format file.

Transfer data is collected during entity processing. This data is encoded and placed into the Exchange Format file.

Error messages or condition codes are sent to the "Run System" to indicate the status of the transfer.

The Post-Processor provides the interface from the Exchange Format to the Working Form.

A set of tables, obtained from the Run System, is used to map the Exchange Format entities to the Working Form physical schema. The Access Software is then used to place these entities into the Working Form.

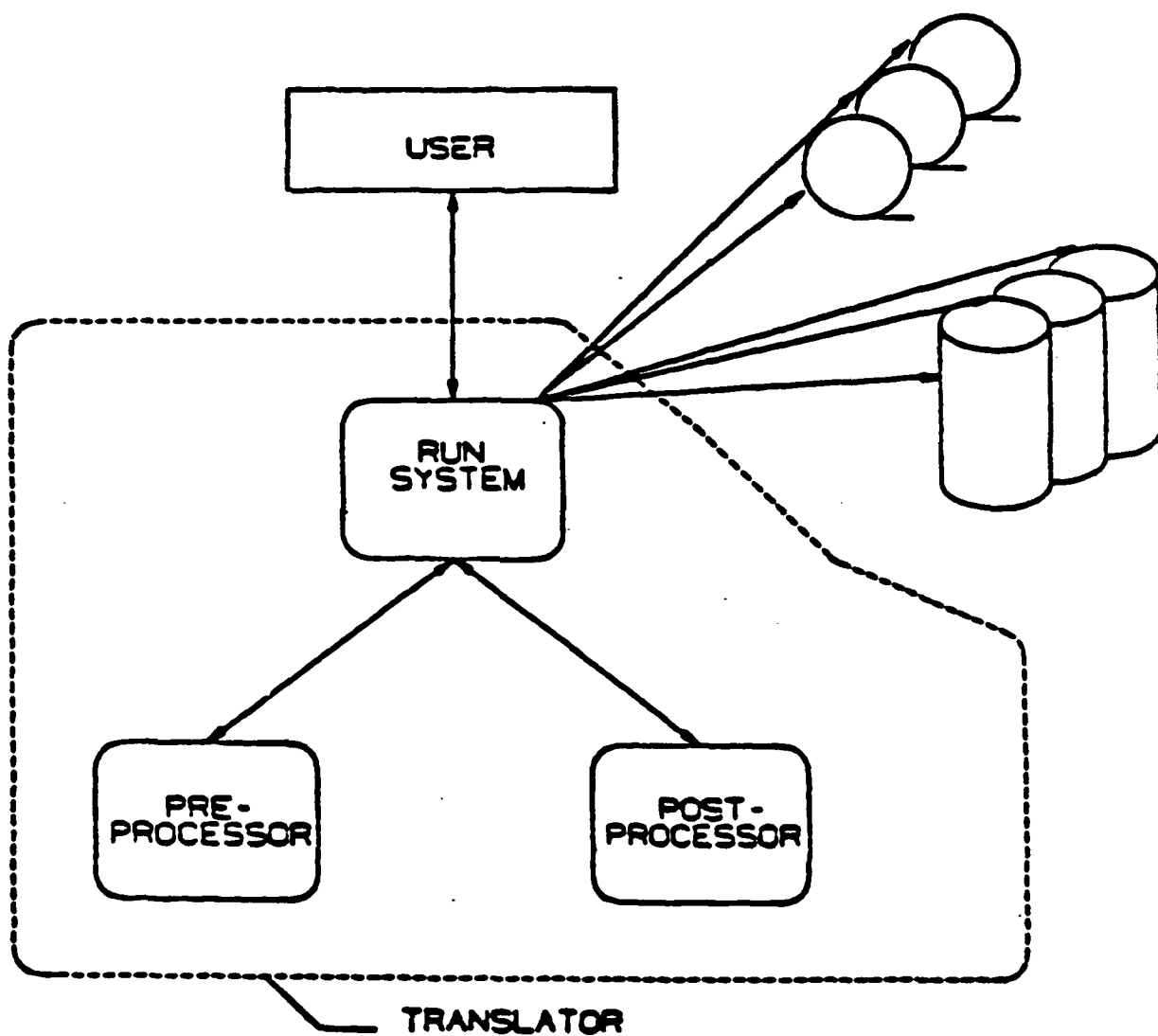


Figure 3-3 Translator Architecture

3.7 Model Access Software

The PDDI Access Software is a set of Pascal procedures that maintains the physical structure of related user data in computer memory. This user data is referred to as the working form model. The package provides an interface to the working form model for application programs to create, relate, and access elements of user data.

The application programs are independent of the physical structure of the stored data elements. This independence ensures that as different structure techniques are implemented, the application programs need not change.

This package manages two types of data: entities and lists. An entity is an element of data supplied by the application to be stored in the working form. A list is a collection of entity keys. The package manages lists created by the application in the working form.

The Access Software allows the structuring of the user data. The entities can be related in user/constituent order. An entity may be related to multiple user entities, creating a network structure in the working form. An entity may also contain multiple constituent entities.

3.8 DATA ITEMS

The Access Software manages two types of data items within the working form - Entities and Lists.

ENTITY

An entity is the principle data item managed by the Access Software, and is:

- o Defined by the conceptual schema in the application creating the entity.
- o Accessed by a unique key return from the create entity function
- o A node in the working form structure containing an Attribute Data Block(ADB), and references to other entities in Constituent Relationships and/or User Relationships

ATTRIBUTE DATA BLOCK

An Attribute Data Block(ADB) is a collection of data embedded in a single contiguous block of memory. Individual pieces of data within an ADB are call attributes. MAS manages only the first three items in the structure of an ADB. These three attributes, KIND, LENGTH, and SYSUSE, are required in every entity. A short description of each attribute follows:

KIND - Must be the first item defined in the ADB. The KIND defines the entity type code. This code cannot be changed.

LENGTH - Must be the second item defined in the ADB. The LENGTH defines the number of bytes in the ADB including KIND, LENGTH, and SYSUSE.

SYSUSE - One full word of system use data reserved for internal purposes. This data is never used by the application, and should never be inspected or modified.

NOTE: All other attribute data in the ADB is managed by the application program.

CONSTITUENT RELATIONSHIP

A constituent entity is used in the definition of the user entity. Inclusive constituents of an entity encompass all descendants, their descendants, and so forth until there are no more descendants. For example in Figure 3-4, Point 0 (P0) and Point 1 (P1) are constituents of Line 1.

```

LINE = ENTITY(5008);
  IDENT : KEY T_IDENT;
  DISPLA : T_DISPLAY;
  P0 : POINT;
  P1 : POINT;
END_ENTITY;
  
```

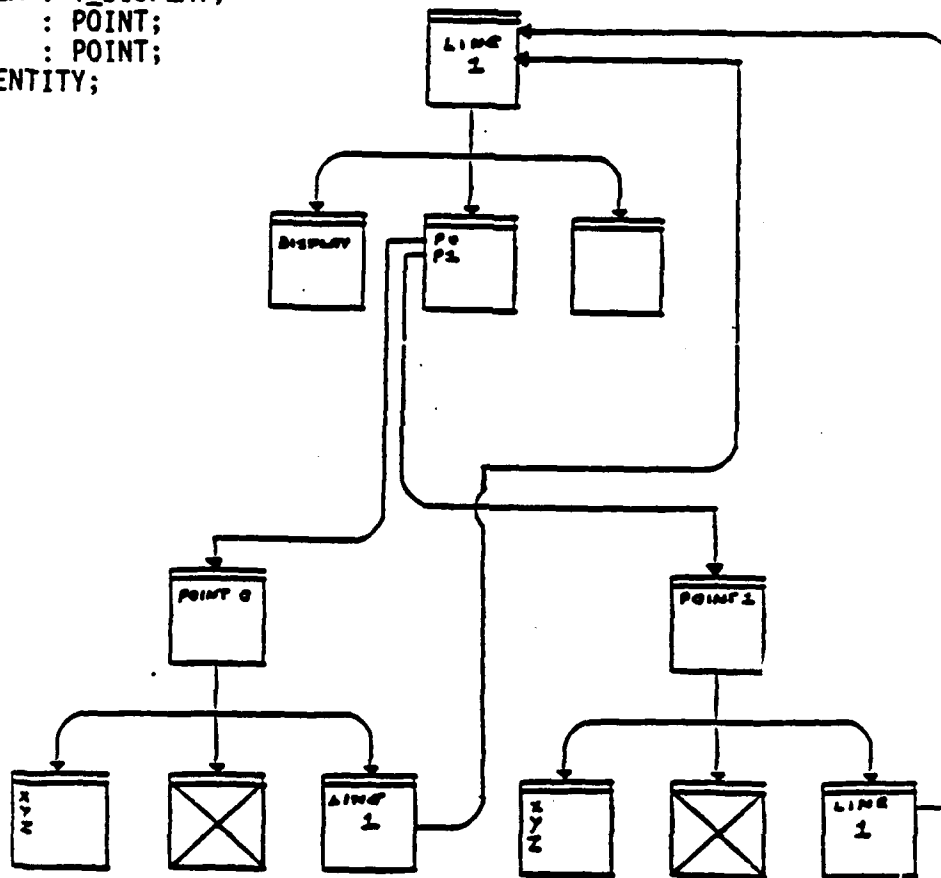


Figure 3-4 LINE: An Entity With Constituents

USER RELATIONSHIPS

A user entity uses constituent entities in its definition. Inclusive users of an entity include all ancestors, their ancestors, and so forth until there are no more ancestors. For example in Figure 1-1, Line 1 is a user of Point 0 (P0) and Point 1 (P1).

LIST

A list is a collection of entity keys which is:

- o Created by the Application program.
- o Accessed by a unique list key returned from the Create List Functions.
- o Used by the Application to store selected entity keys for subsequent processing.

3.9 INTERFACE PARAMETERS

Each interface parameter has a name and a type. This information is shown as follows:

DATA-NAME:DATA-TYPE.

DATA-NAME PARAMETERS

The following conventions are used to name parameters:

- Keys are named KEY1, KEY2,...KEYN.
- The ADB is named ENTDEF.
- Text parameters are named according to their purpose.
- Integer parameters are named according to their purpose.
- A return code is produced by every interface routine/operation. This parameter is a full word integer and is always named IRC. (See Appendix for a return code list.)

DATA-TYPE PARAMETERS

Data-Type parameters may be one of the following:

ANYKEY - Access key of an entity or list.

ENTBLOCK - Entity data block definition.

- In Pascal, probably declared as a record.
- In Fortran, declared as a common or dimension array.

CHARACTER - A single character as defined by the system.

INTEGER - A full word integer.

FORMAL DATA TYPES

The following is a reference list of data-types for interface calls in this MAS document.

ANYKEY	= INTEGER
LISTKEY	= ANYKEY
ENTKEY	= ANYKEY
ORD_KIND	= INTEGER
EXT_RET_CODE	= INTEGER
LISTPSTN	= INTEGER
LISTINIX	= INTEGER
LISTSIZE	= INTEGER
ROUTINE	= ARRAY(1...8) OF CHARACTER
NAMTYP	= ARRAY(1...6) OF CHARACTER
(ADB) ENTBLOCK	= RECORD OF
KIND	= ORD_KIND
SIZE	= INTEGER
SYSUSE	= INTEGER
DATA	= (USER DEFINED)

PASCAL APPLICATIONS

The formal declarations for the Access Software interface routines are maintained in the member APL TYP of the library "CAD5.FRMI.MASymmdd.INCLD"

Where:

y = year
mm = month
dd = day

of the latest Access Software release.

UM 560130001
1 January 1987

3.10 MEMORY MANAGER

A Model Access Memory Manager was developed to replace the PASCAL run-time memory manager. It reduces the number of bytes of overhead required for free-space collection, and isolates the working form model from all other PASCAL dynamic allocations.

This memory manager is currently in the MAS package and requires no user intervention for utilization.

SECTION 4
INTERFACE OPERATIONS

	<u>Page</u>
INTRODUCTION	4-3
INITIALIZATION/DELETION OF THE MAS ENVIRONMENT	4-4
NAME	4-5
NAME	4-6
ENTITY OPERATIONS	4-7
CREATE OPERATIONS	4-8
NAME	4-9
NAME	4-10
NAME	4-11
NAME	4-12
NAME	4-13
NAME	4-14
NAME	4-15
GET OPERATIONS	4-16
NAME	4-17
NAME	4-18
UPDATE OPERATIONS	4-19
NAME	4-20
DELETE OPERATIONS	4-21
NAME	4-24
NAME	4-25
NAME	4-26
NAME	4-27
NAME	4-28
ACTIVATE OPERATIONS	4-29
NAME	4-30
NAME	4-31
NAME	4-32
APPLICATION FLAG OPERATIONS	4-33
NAME	4-34
NAME	4-35
NAME	4-36
NAME	4-37
NAME	4-38
NAME	4-39
NAME	4-40
NAME	4-41

INTERFACE OPERATIONS (CONTINUED)

	<u>Page</u>
LIST OPERATIONS	4-42
BOOLEAN OPERATIONS	4-43
MALAND.	4-44
MALNOT.	4-45
MALOR	4-46
STRUCTURE OPERATIONS	4-47
MALK.	4-48
MALKL	4-49
GENERAL OPERATIONS	4-50
MAL	4-51
MALN.	4-52
MALCPY.	4-53
MALFND.	4-54
MALNO	4-55
MALGTK.	4-56
DELETE OPERATIONS.	4-57
MALD.	4-58
MALDA	4-59
MALDI	4-60
MALOCK.	4-61
EDIT OPERATIONS.	4-62
MALATC.	4-63
MALINS.	4-64
MALRDE.	4-65
MALREP.	4-66
MALRMV.	4-67
MALROR.	4-68
MALRPL.	4-69
MALRVS.	4-70
MALSRT.	4-71
SEQUENTIAL READ AND EXECUTE OPERATIONS	4-72
MALRD	4-73
MALSTF.	4-74
MALSTR.	4-75
MAEXEQ.	4-77
MAKXEQ.	4-78
MALXEQ.	4-79
MAECXQ.	4-81
MAEUXQ.	4-82

4.1 INTRODUCTION

The Entity Operations and List Operations sections provide the applications programmer with the interface operations needed to access the data structures passed back to the application program. (See appendix for Pascal and FORTRAN Schema Diagrams.)

Figure 4-1 illustrates the interrelationships of the Access Software interface operations shown in these sections.

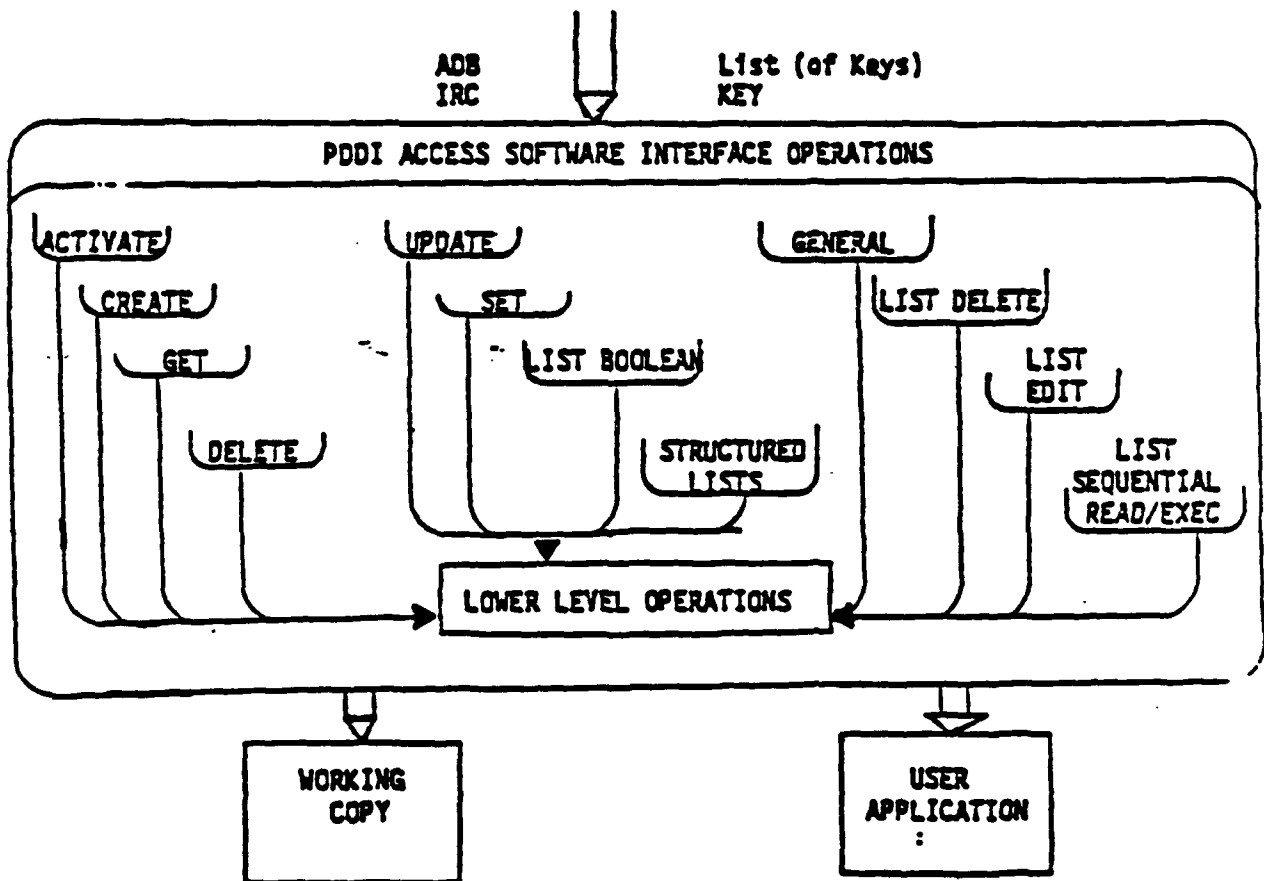


Figure 4-1 Interface Operations

4.2 INITIALIZATION/DELETION OF THE MAS ENVIRONMENT

Two routines provide the interface used to initialize the Access Software.

The basic initialization operation (MAINIT) creates a working model and enables the Access Software.

The MAKILL function is used to destroy the working model and disable the Access Software.

An application does not have to install a data dictionary. It can create and use entities on an ad hoc basis. If a data dictionary is not installed, the following limitations are imposed:

1. Any entity type will be permitted.
2. The interface routines will not validate any operation other than outright errors; e.g., defining an ADB with a negative length. The application is - "on its own".
3. There will be no provision for organization of entities by class.

Included with the initialization and deletion operations descriptions that follow are the error and warning messages that may be returned. Appendix A contains a complete list of these messages along with their numeric codes.

MAINIT

FUNCTION: Initialize the working model.

FORMAT: MAINIT (IRC)

INPUT:
None

OUTPUT:
IRC: INTEGER
The procedure return code.

DESCRIPTION: The working model will be initialized.

The Access Software is enabled.

ERRORS:	<u>Message</u>	<u>Explanation</u>
	MAS_INIT_FAILED	Could not create schema and its root.
	MAINIT_ALREADY_DONE	Root already created.
	NOT_ENOUGH_ROOM	No more core memory.

NOTE: Do not call MAINIT twice in succession. This will result in 2 Access Software environments. Use a MAKILL to delete the current environment before initializing another.

MAKILL

FUNCTION: Delete the current working model.

FORMAT: MAKILL (IRC)

INPUT:
None

OUTPUT:
IRC: INTEGER
The procedure return code.

DESCRIPTION: The entire working model is destroyed.
The Access Software is disabled.

ERRORS: None.

4.3 ENTITY OPERATIONS

The basic entity operations can be categorized by the following functions:

- Activate
- Create
- Get
- Delete
- Update
- Process Flags
- Application Flags

All operations performed on entity constituent lists are done by list operations, with the exception of creating an entity with constituents.

Included with the entity operations descriptions presented on the pages that follow are the error and warning messages that may be returned. Appendix A contains a complete list of these messages along with their numeric codes.

4.3.1 CREATE OPERATIONS

These operations allow the creation of entities in the working model. The application creates the entity in its local memory space. This includes the specification of KIND, LENGTH, and any other attribute data as needed. The KIND value cannot change. The LENGTH value can be changed by the MAEUD function.

The create routines are shown in the following table.

DESCRIPTION	ROUTINE
Create an entity.	MAECR
Create an application list of constituent entity references.	MAEC
Create an application list of inclusive constituent entities.	MAECI
Create an application list of inclusive constituents by KIND.	MAECIK
Create an application list of user entity references.	MAEU
Create an application list of inclusive user entities.	MAEUI
Create an application list of inclusive users by KIND.	MAEUIK

MAECR

FUNCTION: Create an entity.

FORMAT: MAECR(ENTDEF,KEY1,KEY2,IRC)

INPUT:

ENTDEF : ENTBLOCK
The application data structure which contains
the entity definition.

KEY1 : ANYKEY
The entity or list of entities to be made
constituents of the entity being created.

OUTPUT:

KEY2 : ENTKEY
The key of the newly created entity.

IRC : INTEGER
The return code.

DESCRIPTION: The entity is added to the model. Constituent entities are connected to the entity. If KEY1 is an entity key then only that entity will become a constituent. If KEY1 is a list key then all entities in the list will become constituents.

A nil key may be used if the entity being created is to have no constituents(a full word integer zero can be used as a nil key).

NOTE: The application is responsible for the format of the AC data after the first three items (KIND, SIZE, SYSUSE).

EXAMPLE: See Sample Programs Section.

ERRORS:

Message

Description

BAD_LIST_POSITION
BAD_ENT_KIND
NOT_ENOUGH_ROOM

Incorrectly built model.
Kind of given key undefined.
No more core memory.

MAEC

FUNCTION: Create an application list of constituent entities.

FORMAT: MAEC(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities for which a list
of direct constituents is wanted.

OUTPUT:

KEY2 : LISTKEY
The returned key of the application list of
direct constituents.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key then the constituent
list of KEY1 will be copied into KEY2. If KEY1 is a list key
then the constituent lists of each entity will be copied into
KEY2.

ERRORS:

Message

Description

BAD_LIST_REFERENCE
BAD_ENT_KEY
NOT_ENOUGH_ROOM

Given key not an entity or a list.
Nil key.
No more core memory.

MAECI

FUNCTION: Create an application list of inclusive constituent entities.

FORMAT: MAECI(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose inclusive constituents are wanted.

OUTPUT:

KEY2 : LISTKEY
The returned key of the inclusive application list of constituents.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key, then the inclusive constituent list of KEY1 will be copied into KEY2. If KEY1 is a list key, then the inclusive constituent lists of each entity will be copied into KEY2. KEY1 is not included in KEY2.

No duplicate keys will exist. Entities are marked as "processed" when placed in the output list. If a processed entity is encountered again on another constituent list, it will not be repeated on the output list.

EXAMPLE: See Sample Programs Section.

NOTE: See the System Overview Section for further explanation of inclusive constituents.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	BAD_ENT_KEY	Nil key.
	NO_MORE_ROOM	No more core memory.

WARNING: NO_LIST_CREATED

MAECIK

FUNCTION: Create an application list of inclusive constituents of a specified KIND.

FORMAT: MAECIK(KEY1,KIND,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose inclusive constituents are to be searched for by specified KIND.

KIND : INTEGER
The KIND code of an entity or an entity class.

OUTPUT:

KEY2 : LISTKEY
The key of a list which will contain all entities of the specified KIND found within the inclusive constituents of KEY1.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key then the inclusive constituents of the specified KIND will be copied into KEY2. If KEY1 is a list key then the inclusive constituents of all entities on the list of the specified KIND will be copied into KEY2.

No duplicate keys will exist.

NOTE: See Entity in the System Overview for further explanation of inclusive constituents.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	BAD_ENT_KEY	Nil key.
	NO_MORE_ROOM	No more core memory.
WARNING:	NO_LIST_CREATED	

MAEU

FUNCTION: Create an application list of user entity references.

FORMAT: MAEU(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities for which a list
of direct users is wanted.

OUTPUT:

KEY2 : LISTKEY
Returned key of the application list of direct
users.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key then the user list of
KEY1 will be copied into KEY2. If KEY1 is a list key then the
user lists of each entity will be copied into KEY2.

EXAMPLE: See Sample Program Section.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_ENT_KEY	Nil key.
	NO_MORE_ROOM	No more core memory.
WARNING:	NO_LIST_CREATED	

MAEUI

FUNCTION: Create an application list of inclusive user entities.

FORMAT: MAEUI(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose inclusive users are wanted.

OUTPUT:

KEY2 : LISTKEY
The returned key of the inclusive application list of users.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key, then the inclusive user list of KEY1 will be copied into KEY2. If KEY1 is a list key, then the inclusive user lists of each entity will be copied into KEY2. KEY1 is not included in KEY2.

No duplicate keys will exist.

NOTE: See the System Overview Section for further explanation of inclusive users.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	BAD_ENT_KEY	Nil key.
	NO_MORE_ROOM	No more core memory.

WARNING: NO_LIST_CREATED

MAEUIK

FUNCTION: Create an application list of inclusive users by KIND.

FORMAT: MAEUIK(KEY1,KIND,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose inclusive users are to be searched for by specified KIND.

KIND : INTEGER
The KIND code of an entity or an entity class.

OUTPUT:

KEY2 : LISTKEY
The key of a list which will contain all entities of the given KIND found within the inclusive users of KEY1.

IRC : INTEGER
The return code.

DESCRIPTION: KEY2 is created. If KEY1 is an entity key, then the inclusive users of the specified KIND will be copied into KEY2. If KEY1 is a list key, then the inclusive users of all entities on the list of the specified KIND will be copied into KEY2.

NOTE: See the System Overview Section for further explanation of inclusive users.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	BAD_ENT_KEY	Nil key.
	NO_MORE_ROOM	No more core memory.

WARNING: NO_LIST_CREATED

4.3.2 GET OPERATIONS

These operations are used to get the MAS copy of a specified entity attribute block and load it into the application memory area. Get operations are also used to get a specified attribute in the entity ADB.

The get routines are shown in the table below.

DESCRIPTION	ROUTINE
Get the KIND value of a specific entity.	MAEGKN
Get the ADB of a specific entity.	MAEGTK

MAEGKN

FUNCTION: Get the KIND value of a specific entity.

FORMAT: MAEGKN(KEY1,KIND,IRC)

INPUT:

KEY1 : ENTKEY
The entity whose kind is to be gotten.

OUTPUT:

KIND : INTEGER
The KIND value of the specified entity.

IRC : INTEGER
The return code.

DESCRIPTION: The stored ADB is located. The KIND value in the ADB is moved to the application ADB.

ERRORS:

Message

Description

BAD_ENT_KEY
NO_MORE_ROOM

Nil key or not an entity.
No more core memory.

MAEGTK

FUNCTION: Get the ADB of a specific entity.

FORMAT: MAEGTK(KEY1,ENTDEF,IRC)

INPUT:

KEY1 : ENTKEY
The key of the entity to be gotten.

OUTPUT:

ENTDEF : ENTBLOCK
The ADB to receive the stored entity.

IRC : INTEGER
The return code.

DESCRIPTION: The stored ADB is located and moved to the application ADB. If KEY1 is a nil key, then a nil KIND and a zero LENGTH is returned.

EXAMPLE: See Sample Program Section.

ERRORS

Message

Description

BAD_ENT_KEY
NO_MORE_ROOM

Nil key or not an entity.
No more core memory.

4.3.3 UPDATE OPERATIONS

These operations are used to update the ADB for specified entities. In general, the application should use the MAEGTK function to get the ADB before the update function is used.

The update routine is shown in the following table.

DESCRIPTION	ROUTINE
Update the attribute data block of an entity.	MAEUD

MAEUD

FUNCTION: Update the attribute data block of an entity.

FORMAT: MAEUD(KEY1,ENTDEF,IRC)

INPUT:

KEY1 : ENTKEY
The entity to be updated. This must be an
entity key.

ENTDEF : ADB
The ADB supplying the update values.

OUTPUT:

IRC : INTEGER
The return code.

DESCRIPTION: The ADB of KEY1 will be updated. The value of KIND must agree with the Model Access Software copy of this entity. Otherwise, an error will result. If the LENGTH is greater than the current LENGTH, then a new ADB will be created with more space.

ERRORS:

<u>Message</u>	<u>Description</u>
BAD_ENT_KIND	Kind or given key is undefined.
BAD_ENT_KEY	Given key is nil.
CANT_UPDATE_ENTITY	
NO_MORE_ROOM	No more core memory.

4.3.4 DELETE OPERATIONS

These operations address how you delete entities from the MAS working form model. The entities in the working model currently are grouped into the following classifications:

- o Dependent
- o Support
- o Primary
- o Secondary

Delete rules have been established for the entities in these classifications. For a new entity kind, the default classification is "Dependent" unless it is otherwise defined.

DELETE RULES

The delete rules apply to the constituent relationships with which entities are defined. They determine whether a constituent entity can be deleted by checking each of its user entities. For example, the delete rules applied to entity (A) in relation to a specific user entity (B) may be different than the delete rules for that same entity (A) in relation to another specific user entity (C).

The action taken for the IDB/MAS delete classifications are determined by the combinations of yes/no (Y/N) answers to the following conditions, posed as questions:

- (1) Can this constituent entity be deleted from a specific user entity?
- (2) Does the deletion of this (constituent) entity cause deletion of a specific user?
- (3) Does deletion of a specific user cause deletion of this entity (constituent)?

CONDITION			DELETE CLASSIFICATION
(1)	(2)	(3)	
N	N	N	Dependent
N	N	Y	Support
N	Y	N	Primary
Y	N	N	Secondary

The delete classifications are defined as follows:

- Dependent - Constituent entity cannot be deleted because the user entity is dependent on its existence. The user entity may be deleted without deleting the constituent entity.
- Support - Constituent entity cannot be deleted because the user entity is dependent on its existence. The user entity may be deleted; however, the constituent entity will also be deleted unless another user entity does not permit the deletion of the constituent entity.
- Primary - Constituent entity can be deleted, but only if the user entity can, and will, also be deleted. The user entity may be deleted without the constituent entity being deleted.
- Secondary - If the number of constituents falls below an established minimum, the constituent entity can be deleted and, if possible, the user entity will also be deleted. If the user entity cannot be deleted, none of the minimum constituents can be deleted. If the number of constituents is greater than or equal to the minimum, the constituent entity can be deleted.

Test routines are provided to return the entities or lists that would be deleted if actual delete routines were used.

DELETE ROUTINES

The IDB/MAS delete routines are presented in the following table. The first two routines actually delete entities (MAED, MAEDI). The third and fourth routines test the delete function, allowing the programmer to see the results of a potential delete without modifying the stored data (MAEDT, MAEDTI).

When deleting a list of entities that includes users and constituents, the list should be ordered so that the users are processed before the constituents. The routines MALROR and MALRORI perform this function on an application list. (An entity constituent list should never be reordered.)

DESCRIPTION	ROUTINE
Delete an entity or list of entities.	MAED
Delete an entity or list of entities and the inclusive constituents.	MAEDI
Delete test an entity or list of entities.	MAEDT
Delete test an entity or list of entities and the inclusive constituents.	MAEDTI

MAED

FUNCTION: Delete an entity or list of entities.

FORMAT: MAED(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be deleted.

OUTPUT:

KEY2 : LISTKEY
The list of entities marked for deletion.

IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key, and the order of the entities in the list may be important. KEY2 will list any entities from the KEY1 list that were not deleted. If all entities are deleted, the mark list will be empty.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	RULE_DOES_NOT_MATCH	Rules defined incorrectly.
	NO_MORE_ROOM	No more core memory.
WARNINGS:	EMPTY_MARK_LIST	No entities marked for delete.

MAEDI

FUNCTION: Delete an entity or list of entities and their inclusive constituents.

FORMAT: MAEDI(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be deleted.

OUTPUT:

KEY2 : LISTKEY
The list of entities marked for delete.
IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key, and the order of the entities in the list may be important. KEY2 will list any entities from the KEY1 list that were not deleted. If all entities are deleted, the mark list will be empty.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	RULE_DOES_NOT_MATCH	Rules defined incorrectly.
	NO_MORE_ROOM	No more core memory.
WARNINGS:	EMPTY_MARK_LIST	No entities marked for delete.

MAEDT

FUNCTION: Delete test an entity or list of entities.

FORMAT: MAEDT(KEY1,KEY2,KEY3,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be tested.

OUTPUT:

KEY2 : LISTKEY
The list containing entities that would be
deleted by MAED.

KEY3 : LISTKEY
The list containing entities that would be
marked by MAED.

IRC : INTEGER
The return code.

DESCRIPTION: The MAEDT delete routine simulates the output of the MAED routine without actually deleting the entities or marking them inactive.

ERRORS:

Message

Description

BAD_DELETE_KEY
RULE_DOES_NOT_MATCH
NO_MORE_ROOM

Given key not an entity or a list.
Rules defined incorrectly.
No more core memory.

WARNINGS:

EMPTY_DELETE_LIST
EMPTY_MARK_LIST

Given delete list was empty.
No entities marked for delete.

MAEDTI

FUNCTION: Delete test an entity or list of entities and their inclusive constituents.

FORMAT: MAEDTI(KEY1,KEY2,KEY3,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be tested.

OUTPUT:

KEY2 : LISTKEY
The list containing entities that would be deleted by MAEDI.

KEY3 : LISTKEY
The list containing entities that would be marked by MAEDI.

IRC : INTEGER
The return code.

DESCRIPTION: The MAEDTI delete routine simulates the output of the MAEDI routine without actually deleting the entities or rendering them inactive.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key not an entity or a list.
	BAD_DELETE_KEY	Given key is nil.
	RULE_DOES_NOT_MATCH	Rules defined incorrectly.
	NO_MORE_ROOM	No more core memory.
WARNINGS:	EMPTY_DELETE_LIST	Given delete list was empty.
	EMPTY_MARK_LIST	No entities marked for delete.

MAEDTS

FUNCTION: Delete test an entity or list of entities and return three lists.

FORMAT: MAEDTS(KEY1,KEY2,KEY3,KEY4,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be tested.

OUTPUT:

KEY2 : LISTKEY
The list of entities that would be deleted by MAED.

KEY3 : LISTKEY
The list of entities that would not be deleted by MAED.

KEY4 : LISTKEY
The list of entities that would be marked for delete by MAED.

IRC : INTEGER
The return code.

DESCRIPTION: The MAEDTS routine is similar to MAEDT except that three lists are returned. KEY2 and KEY4 can be submitted to directly delete and mark entities without checking the delete rules.

ERRORS:

Message

Description

BAD_LIST_REFERENCE
BAD_DELETE_KEY
RULE_DOES_NOT_MATCH
NO_MORE_ROOM

Given key not an entity or a list.
Given key is nil.
Rules defined incorrectly.
No more core memory.

WARNINGS:

EMPTY_DELETE_LIST
EMPTY_MARK_LIST

Given delete list was empty.
No entities marked for delete.

4.3.5 ACTIVATE OPERATIONS

These operations are used to activate an entity. An entity is deactivated when a delete was attempted, but was not completed because of the user's dependency condition on the entity. (See Delete Operations Section)

The activate routines are shown in the table below.

DESCRIPTION	ROUTINE
Activate an entity or list of entities.	MAEA
Activate an entity or list of entities and their inclusive constituents.	MAEAI
Find the present value of the activation setting for an entity.	MAEAV

NOTE:

- o Activation is not the same as rejection after a delete. If an entity was deleted, then it cannot be recovered with these functions.
- o Activation functions will activate any entity regardless of when or how it was made inactive.

UM 560130001
1 January 1987

MAEA

FUNCTION: Activate an entity or list of entities.

FORMAT: MAEA (KEY1,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be activated.

OUTPUT:

IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key then only that entity will be activated. If KEY1 is a list key then all entities in the list will be activated.

ERRORS:

Message

Explanation

BAD_LIST_REFERENCE

Given key not an entity or a list.

BAD_ENT_KEY

Nil key.

NOT_ENOUGH_ROOM

No more core memory.

MAEAI

FUNCTION: Activate an entity or list of entities and their inclusive constituents.

FORMAT: MAEAI(KEY1,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities to be activated.

OUTPUT:

IRC : INTEGER
The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key then only that entity and its inclusive constituents will be activated. If KEY1 is a list key then all entities in the list and their inclusive constituents will be activated.

See the System Overview Section for further explanation of inclusive constituents.

ERRORS:

Message

Description

BAD_LIST_REFERENCE
NOT_ENOUGH_ROOM

Given key not an entity or a list.
No more core memory.

MAEAV

FUNCTION: Find the present value of the activation setting for an entity.

FORMAT: MAEAV(KEY1,IAVAL,IRC)

INPUT:

KEY1 : ENTKEY
The entity to be examined.

OUTPUT:

IAVAL : INTEGER
The activation code.
= 0 if set for delete
= 1 if not set for delete

IRC : INTEGER
The return code.

DESCRIPTION: Returns the current value of the activation setting for the specified entity.

ERRORS:

Message

Description

BAD_ENT_KEY
NOT_ENOUGH_ROOM

Given key not an entity.
No more core memory.

4.3.6 APPLICATION FLAG OPERATIONS

These operations are used to get or set any application accessible flag associated with an entity.

The Application Flag routines are shown in the following table.

DESCRIPTION	ROUTINE
Reset any application accessible flag for all entities in the model.	MAERST
Determine the value of a given application accessible flag of an entity.	MAQURY
Update the value of a given application accessible flag of an entity or list of entities.	MAUPDT
Determine whether the user compresses with its constituent.	MAECQY
Create a list of constituents with which the input entity compresses.	MAECMP
Reset Process Flag for all entities in the model.	MAESWA
Set the Process Flag in an entity or list of entities.	MAESWT
Find the Process Flag setting of an entity.	MAESVL

MAERST

FUNCTION: Reset given application accessible flag in all entities in the model.

FORMAT: MAERST(FLAGSNAME,IRC)

INPUT:

FLAGSNAME : NAMTYP

The name of the flag to be reset in all entities in the model. Can have the following values:

- 1) 'MRDFLG' activation flag
 - 2) 'PRCFLG' process flag
 - 3) 'ABSFLG' absent/present flag
 - 4) 'APLFLG' application flag
- 0 off
-1 on

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: Determine what flag is to be reset in every entity in the model. Resets that flag to 'off'.

ERRORS:

Message

Description

INVALID_FLAG_NAME
SCHEMA_ROOT_NIL
NO_MORE_ROOM

Given flag name undefined.
No model active.
No more core memory.

MAQURY

FUNCTION: Determine the value of a given application accessible flag for the entity.

FORMAT: MAQURY(KEY1,FLAGNAME,FLGVAL,IRC)

INPUT:

KEY1 : ENTKEY
The entity whose specified flag value is to be gotten.

FLAGNAME : NAMTYP
The name of the flag to be gotten. Can have the following values:

- 1) 'MRDFLG' activation flag
 - 2) 'PRCFLG' process flag
 - 3) 'ABSFLG' absent/present flag
 - 4) 'APLFLG' application flag
- =0 off
=1 on

OUTPUT:

FLGVAL : INTEGER
The value of the specified flag.
0 = false
1 = true

IRC : INTEGER
The return code.

DESCRIPTION: Determine what flag's value is to be gotten. Get that flag's value.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given reference nil or not an entity.

INVALID_FLAG_NAME

Given flag name undefined.

NO_MORE_ROOM

No more core memory.

MAUPDT

FUNCTION: Update the value of a given application accessible flag for an entity or list of entities.

FORMAT: MAUPDT(KEY1,FLGNAME,FLGVAL,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose specified flag value will be updated.

FLAGNAME : NAMTYP
The name of the flag to be updated. Can have the following values:

- 1) 'MRDFLG' activation flag
- 2) 'PRCFLG' process flag
- 3) 'ABSFLG' absent/present flag
- 4) 'APLFLG' application flag
 =0 off
 =1 on

FLGVAL : INTEGER
The value of the specified flag to be used when updating.
0 = false
1 = true

IRC : INTEGER
The reutrn code.

DESCRIPTION: Determine what flag's value is to be updated. Update that flag's value.

ERRORS:

Message

Description

BAD_LIST_REFERENCE
INVALID_FLAG_NAME
NO_MORE_ROOM

Given key not an entity or a list.
Given flag name undefined.
No more core memory.

MAECQY

FUNCTION: Determine whether the user compresses with its constituent.

FORMAT: MAECQY(KEY1, KEY2, CMPFLG, IRC)

INPUT:

KEY1 : ENTKEY
Key of the entity thats constituent is to be checked.

KEY2 : ENTKEY
Key of the constituent thats rule is to be checked.

OUTPUT:

CMPFLG : INTEGER
Value of the user's compress rule in relation to its constituent.
1 = true
0 = false

IRC : INTEGER
Return code
0 = Good return
<0 Critical error
>0 Warning

DESCRIPTION: Query constituent compress rule to its user.

MAECMP

FUNCTION: Create a list of constituents with which the input entity compresses.

FORMAT: MAECMP(KEY1, KEY2, IRC)

INPUT:

KEY1 : ENTKEY
Key of the entity that its compressibility is determined by the constituents(s).

OUTPUT:

KEY2 : LISTKEY
List of the constituents that cause the compression of the input entity.

IRC : INTEGER
Return code
0 = Good return
<0 Critical error
>0 Warning

DESCRIPTION: Each constituent that delete rule states that the input entity will also be compressed will be added to the output list.

UM 560130001
1 January 1987

MAESWA

FUNCTION: Reset Process Flag for all entities in the model.

FORMAT: MAESWA(IRC)

INPUT:
NONE

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: The Process Flag is set to OFF in all entities in the working-form model.

ERRORS:	<u>Message</u>	<u>Description</u>
	NO_MORE_ROOM	No more core memory.

MAESWT

FUNCTION: Set the Process Flag in an entity or a list of entities.

FORMAT: MAESWT(KEY1,ISWT,IRC)

INPUT:

KEY1 : ANYKEY
The entity or list of entities whose process flag is to be set.

ISWT : INTEGER
The input value of the process flag.

OUTPUT:

IRC : INTEGER
The return code.

DESCRIPTION: The process flag will be set to the value specified by ISWT. If KEY1 is an entity key, then the flag in that entity will be set. If KEY1 is a list key, then the flag in all entities referenced by the list will be set. ISWT should be "1" for flag setting of true and "0" for flag setting of false.

ERRORS:

Message

Description

BAD_LIST_REFERENCE
NO_MORE_POOM

Given key not an entity or a list.
No more core memory.

MAESVL

FUNCTION: Find the Process Flag setting of an entity.

FORMAT: MAESVL(KEY1,ISET,IRC)

INPUT:

KEY1 : KEY
The entity for which the flag setting is
wanted. This must be an entity key.

OUTPUT:

ISET : INTEGER
The output value of the process flag.

IRC : INTEGER
The return code.

DESCRIPTION: The value of the process flag for KEY1 will be returned. If the
flag is true, then the value "1" will be returned. If the flag
is false, then the value "0" will be returned.

ERRORS:

Message

Description

BAD_ENT_KEY
NO_MORE_ROOM

Given key is nil or not an entity.
No more core memory.

4.4 LIST OPERATIONS

This section explains the use of the MAS list operations. A list is a temporary internal structure that contains references to entities. Since the application can build lists that take up space in the working model, it is necessary that the applications periodically delete the lists that are no longer needed.

Many list operations will accept either a list key or an entity key as input keys. When an entity key is supplied, it is assumed that the constituent list of the entity becomes the list to be operated on.

Some operations on lists may result in the same entity being in the output list more than once. The operation (MALRDE) can be used to remove duplicate entities from the list.

All operations that create an application list automatically set the position of the list to the beginning. Once a list has been read to the end, it must be reset before the sequential read process can begin again.

When an entity is deleted, all references to it in all application lists are automatically removed and the current positions of the affected lists are adjusted to retain their original meaning.

The basic list operations can be categorized by the following functions:

- Boolean
- Structure
- General
- Delete
- Edit
- Sequential Read
- Execute

Included with the list operations descriptions are the error and warning messages that may be returned. Appendix A contains a complete list of these messages along with their numeric codes.

4.4.1 BOOLEAN OPERATIONS

For Boolean operations, there are two input lists and one output list. The application is responsible for providing two input lists consisting with the Boolean operation to be performed. No validation checking is done. If one or both of the input lists contain duplicate entities, then the output list may also contain duplicate entities. This result may not be consistent with the Boolean theory operation being performed.

The Boolean routines are shown in the following table.

DESCRIPTION	ROUTINE
Create a list from a Boolean "AND" on two input lists.	MALAND
Create a list from a Boolean "NOT" on two input lists.	MALNOT
Create a list from a Boolean "OR" on two input lists.	MALOR

MALAND

FUNCTION: Create a list from a Boolean "AND" on two input lists.

FORMAT: MALAND(KEY1,KEY2,KEY3,IRC)

INPUT:

KEY1 : ANYKEY

An entity or a list that is to be AND'ed.

KEY2 : ANYKEY

An entity or a list that is to be AND'ed.

OUTPUT:

KEY3 : LISTKEY

The list of entities that occurred in both KEY1 and KEY2.

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key, then the constituent list of KEY1 is AND'ed with KEY2. If KEY1 is a list key, then KEY1 is AND'ed with KEY2. KEY2 may be either an entity key or a list key. If KEY2 is an entity key then the constituent list of KEY2 is AND'ed with KEY1. If KEY2 is a list key then KEY2 is AND'ed with KEY2. The list KEY3 is created, corresponding to the set theoretical intersection of KEY1 and KEY2.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key(s) nil or not a list.

NO_MORE_ROOM

No more core memory.

WARNINGS:

NO_LIST_CREATED

MALNOT

FUNCTION: Create a list from a Boolean "NOT" on two input lists.

FORMAT: MALNOT(KEY1,KEY2,KEY3,IRC)

INPUT:

KEY1 : ANYKEY

An entity or a list that is to be NOT'ed.

KEY2 : ANYKEY

An entity or a list that is to be NOT'ed.

OUTPUT:

KEY3 : LISTKEY

The list of entities that occurred in KEY1 but not in KEY2.

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key, then the constituent list of KEY1 is NOT'ed with KEY2. If KEY1 is a list key, then KEY1 is NOT'ed with KEY2. KEY2 may be either an entity key or a list key. If KEY2 is an entity key, then the constituent list of KEY2 is NOT'ed with KEY1. If KEY2 is a list key, then KEY2 is NOT'ed with KEY1. The list KEY3 is created, corresponding to the set theoretical difference of KEY1 and KEY2.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key not an entity or a list.

BAD_ENT_KEY

Given key nil.

NO_MORE_ROOM

No more core memory.

WARNINGS:

NO_LIST_CREATED

MALOR

FUNCTION: Create a list from a Boolean "OR" on two input lists.

FORMAT: MALOR(KEY1,KEY2,KEY3,IRC)

INPUT:

KEY1 : ANYKEY

An entity or a list that is to be OR'ed.

KEY2 : ANYKEY

An entity or a list that is to be OR'ed.

OUTPUT:

KEY3 : LISTKEY

The list of entities that occurred in either KEY1 or KEY2.

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity key, then the constituent list of KEY1 is OR'ed with KEY2. If KEY1 is a list key, then KEY1 is OR'ed with KEY2. KEY2 may be either an entity key or a list key. If KEY2 is an entity key, then the constituent list of KEY2 is OR'ed with KEY1. If KEY2 is a list key, then KEY2 is OR'ed with KEY1. The list KEY3 is created, corresponding to the set theoretical union of KEY1 and KEY2. If there is an entity in KEY1 that is also in KEY2, there will be duplicates in KEY3.

ERRORS

Message

Description

BAD_LIST_REFERENCE

Given key not an entity or a list.

BAD_ENT_KEY

Given key(s) nil.

NO_MORE_ROOM

No more core memory.

WARNINGS:

NO_LIST_CREATED

4.4.2 STRUCTURE OPERATIONS

The following table presents the structure routines:

DESCRIPTION	ROUTINE
Create a list of entities with a given KIND.	MALK
Create a list of entities with a given KIND that are found within another list.	MALKL

MALK

FUNCTION: Create a list of entities with a given KIND.

FORMAT: MALK(KIND,KEY1,IRC)

INPUT:

KIND : INTEGER

A valid KIND code that may be either the KIND of an entity or class of entities.

OUTPUT:

KEY1 : LISTKEY

The list of all entities of the specified KIND.

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 is created. KEY1 will contain all entities of KIND. If a data dictionary is specified, then the KIND may be a class of entities. In this case, the elements of the list will be a (logical) concatenation of the content of each entity class as they are encountered from left to right in the entity class structure.

ERRORS:

Message

Description

CANT_CREATE_LIST
BAD_SCHEMA_KIND
BAD_LIST_POSITION
NO_MORE_ROOM

Given kind undefined.
Given not of an instance or class.
Schema root inconsistent.
No more core memory.

WARNINGS:

NO_SUCH_SCHEMA
NO_LIST_CREATED

Kind undefined.

MALKL

FUNCTION: Create a list of entities with a given KIND that are found within another list.

FORMAT: MALKL(KEY1,KIND,KEY2,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities whose list is to be searched for the specified KIND.

KIND : INTEGER

The KIND code of an entity or an entity class.

OUTPUT:

KEY2 : LISTKEY

The list that will contain all entities of the given KIND found within the list specified by KEY1.

IRC : INTEGER

The return code.

DESCRIPTION: If KEY1 is an entity key, put all constituents of entity KEY1 into KEY2 that match on kind. If KEY1 is a list key, put all entities in the KEY1 list into KEY2 that match on kind.

ERRORS:

Message

Description

BAD_ENT_KIND

No such kind.

BAD_LIST_REFERENCE

Given key nil or not an entity or not a list.

BAD_LIST_POSITION

Schema root inconsistent.

NO_MORE_ROOM

No more core memory.

WARNINGS:

NO_LIST_CREATED

4.4.3 GENERAL OPERATIONS

The following table presents the general routines:

DESCRIPTION	ROUTINE
Creates an empty list.	MAL
Create an empty list of specified size.	MALN
Makes a copy of a list.	MALCPY
Find the position of an entity in a list.	MALFND
Count the entities in a list.	MALNO
Get the Nth key from a list.	MALGTK

MAL

FUNCTION: Creates an empty list.

FORMAT: MAL(KEY1,IRC)

INPUT:
None

OUTPUT:
KEY1 : LISTKEY
The key of the empty list.

IRC : INTEGER
The return code.

DESCRIPTION: An empty list is created.

EXAMPLE: See Sample Programs Section.

ERRORS:	<u>Message</u>	<u>Description</u>
	CANT_CREATE_LIST	Memory allocation problems.
	NO_MORE_ROOM	No more core memory.

MALN

FUNCTION: Create an empty list of specified size.

FORMAT: MALN(LSIZE,KEY1,IRC)

INPUT:

LSIZE: INTEGER

The number of entries in the list.

OUTPUT:

KEY1 : LISTKEY

The key of the empty list of specified size.

IRC : INTEGER

The return code.

DESCRIPTION: An empty application list will be created with sufficient space to accommodate LSIZE entries. All entries are initialized to nil.

ERRORS:

Message

Description

CANT_CREATE_LIST
MAXIMUM_LIST_SIZE
NO_MORE_ROOM

Memory allocation problems.
Requested size too large.
No more core memory.

MALCPY

FUNCTION: Makes a copy of a list.

FORMAT: MALCPY(KEY1,KEY2,IRC)

INPUT:

KEY1 : LISTKEY

A list key whose entries will be copied.

OUTPUT:

KEY2 : LISTKEY

The new list that will receive a copy of KEY1.

IRC : INTEGER

The return code.

DESCRIPTION: KEY2 will be created. The elements of KEY1 will be copied into KEY2.

ERRORS:

Message

Description

BAD_LIST_REFERENCE
NO_MORE_ROOM

Given key nil or not a list.
No more core memory.

MALFND

FUNCTION: Find the position of an entity in a list. If KEY1 is an entity then find its position in the constituent list of that entity.

FORMAT: MALFND(KEY1,KEY2,IFIRST,IPOS,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities in which KEY2 is to be found.

KEY2 : ENTKEY

The entity to be found in KEY1.

IFIRST : INTEGER

The position in KEY1 where the find operation is to start.

OUTPUT:

IPOS : INTEGER

The position in KEY1 where KEY2 is found.

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list then KEY2 is found in the list. If KEY1 is an entity, then KEY2 is found in the constituent list of KEY1. KEY2 must be an entity key. The find starts at position IFIRST. Each entity in KEY1 is examined for equality with KEY2 starting with the specified position. If a match is found, then the position is returned in IPOS. If there is no match, then IPOS is returned as zero and IRC signals an error. If there are multiple matches, then only the first (leftmost) match is returned in IPOS.

ERRORS:	<u>Message</u>	<u>Description</u>
	NO_MATCH_FOUND	Entity not on list.
	BAD_LIST_REFERENCE	Given key(s) not an entity or a list.
	BAD_ENT_KEY	Given key(s) is nil.
	NO_MORE_ROOM	No more core memory.

MALNO

FUNCTION: Count the entities in a list.

FORMAT: MALNO(KEY1,KOUNT,IRC)

INPUT:

KEY1 : ENTKEY

The entity or list of entities to be counted

OUTPUT:

KOUNT: INTEGER

The number of entities in the list.

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is an entity the number of constituents is returned. If KEY1 is a list the number of entities on the list is returned.

EXAMPLE: See Sample Programs Section.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key(s) is nil or is not an entity or a list.

NO_MORE_ROOM

No more core memory.

MALGTK

FUNCTION: Get the Nth Key from a list.

FORMAT: MALGTK(KEY1,IPOS,KEY2,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to be processed.

IPOS : INTEGER

The position in the list of the target entity.

OUTPUT:

KEY2 : ENTKEY

The requested key.

IRC : INTEGER

The return code.

DESCRIPTION: If KEY1 is a list, get the IPOS entry from the list. If KEY1 is an entity, get the IPOS entry from the constituent list of KEY1.

EXAMPLE: See Sample Programs Section

ERRORS:

Message

Description

BAD_LIST_POSITION

Given position beyond the length of list.

BAD_LIST_REFERENCE

Given key(s) is nil or is not an entity or a list.

NO_MORE_ROOM

No more core memory.

4.4.4 DELETE OPERATIONS

The following table presents the delete routines:

DESCRIPTION	ROUTINE
Delete an application list.	MALD
Delete all application lists.	MALDA
Delete an application list and all lists created after it.	MALDI
Set or unset the application list lock flag.	MALOCK

MALD

FUNCTION: Delete an application list.

FORMAT: MALD(KEY1,IRC)

INPUT:

KEY1 : LISTKEY

The list to be deleted.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 must not be an entity key. KEY1 is deleted. KEY1 cannot be recovered.

ERRORS:

Message

Description

BAD_DELETE_KEY

Given key not a root, entity, or list.

NO_MORE_ROOM

No more core memory.

MALDA

FUNCTION: Delete all application lists.

FORMAT: MALDA(IRC)

INPUT:
None

OUTPUT:
IRC : INTEGER
The return code.

DESCRIPTION: All application lists will be deleted. They cannot be recovered. If an application list is locked, then it will not be deleted.

ERRORS:

Message

Description

BAD_LIST_POSITION
BAD_LIST_REFERENCE
NO_MORE_ROOM

Error in processing lists.
Found nil list pointer.
No more core memory.

MALDI

FUNCTION: Delete an application list and all lists created after it.

FORMAT: MALDI(KEY1,IRC)

INPUT:

KEY1 : LISTKEY

The list to be deleted.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 must not be an entity key. The list identified by KEY1 and all lists created after it will be deleted. Deleted lists cannot be recovered. If an application list is locked, then it will not be deleted.

ERRORS:

Message

Description

BAD_LIST_POSITION
BAD_LIST_REFERENCE

Error in processing lists.
Given key is nil or not in stack of
lists or is not a list.

BAD_DELETE_LIST
BAD_ENT_KEY
NO_MORE_ROOM

Given key is nil.
Found a nil list.
No more core memory.

MALOCK

FUNCTION: Set or unset the application list lock flag.

FORMAT: MALOCK(KEY1,LOCK,IRC)

INPUT:

KEY1 : LISTKEY
The list to be set.

LOCK : INTEGER
The lock setting
=0 unlocked
=1 locked

OUTPUT:

IRC : INTEGER
The return code

DESCRIPTION: A list that is locked will not be deleted by the interface functions MALDA or MALDI. All other functions that delete lists will delete a locked list.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key is not a list or is nil.
	NO_MORE_ROOM	No more core memory.

4.4.5 EDIT OPERATIONS

The following table presents the edit routines:

DESCRIPTION	ROUTINE
Attach an entity or list of entities to a list.	MALATC
Insert an entity or list of entities into a list.	MALINS
Remove duplicate entries in a list.	MALRDE
Replace a list.	MALREP
Remove an entity from a list.	MALRMV
Reorder list of entities in user to constituent order.	MALROR
Replace an entity in a list.	MALRPL
Reverse the order of a list	MALRVS
Sorts a list using order provided by a user defined function	MALSRT

MALATC

FUNCTION: Attach an entity or list of entities to a list. If KEY1 is an entity then attach to the constituent list of that entity.

FORMAT: MALATC(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to which KEY2 is to be attached.

KEY2 : ANYKEY

The entity or list to be attached to KEY1.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then KEY2 is attached to the list. If KEY1 is an entity, then KEY2 is attached to the constituent list of KEY1. This will make KEY2 a constituent of KEY1. KEY2 may be either an entity key or a list key. If KEY2 is a list, then the entire list is attached to KEY1. This is the same as doing multiple attaches of an entity. If KEY2 is an entity, then the entity is attached to KEY1.

EXAMPLE: See Sample Programs Section.

ERRORS:	<u>Message</u>	<u>Description</u>
	INVALID_CONNECTION	Given key is nil.
	CANT_CONNECT	Given key is not an entity or list.
	BAD_ENT_KEY	Given key(s) is nil.
	NO_MORE_ROOM	No more core memory.

MALINS

FUNCTION: Insert an entity or list of entities into a list. If KEY1 is an entity, then insert into the constituent list of that entity.

FORMAT: MALINS(KEY1,KEY2,IPOS,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities in which KEY2 is to be inserted.

KEY2 : ANYKEY

The entity or list to be inserted in KEY1.

IPOS : INTEGER

The position in KEY1 where the insert is to take place.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then KEY2 is inserted in the list. If KEY1 is an entity, then KEY2 is inserted in the constituent list of KEY1. KEY2 may be either an entity key or a list key. If KEY2 is a list, then the entire list is inserted in KEY1. If KEY2 is an entity, then the entity is inserted in KEY1.

The insert takes place before IPOS. That is, the entity at IPOS is moved by one position if KEY2 is an entity or by the number of elements in the list if KEY2 is a list. Then the elements are moved into the vacated positions.

ERRORS:

Message

Description

BAD_LIST_POSITION

Given position less than 1 or greater than length.

BAD_LIST_REFERENCE

Given key is nil.

INVALID_CRB_POSITION

Pointer to position of last read at constituent list is inconsistent.

CRB_POSITION_NOT_FOUND

No pointer to read position found.

NO_MORE_ROOM

No more core memory.

MALRDE

FUNCTION: Remove duplicate entries in a list.

FORMAT: MALRDE(KEY1,IRC)

INPUT:

KEY1 : LISTKEY
The input/output list.

OUTPUT:

IRC : INTEGER
The return code.

DESCRIPTION: Any duplicate entities found in the input list will be removed. The change is made in-place. The first instance of each entity will be kept.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key is not a list.
	DUPLICATES_NOT_REMOVED	
	NO_MORE_ROOM	No more core memory.

MALREP

FUNCTION: Replace a list. If KEY1 is an entity then replace the constituent list of that entity.

FORMAT: MALREP(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to be replaced.

KEY2 : ANYKEY

The entity or list to replace KEY1.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list then KEY2 replaces KEY1. If KEY1 is an entity then the constituent list of KEY1 is replaced by KEY2. KEY2 may be either an entity or a list key.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key(s) is nil or not an entity or a list.

INVALID_CRB_POSITION

Pointer to position of last read of constituent list is inconsistent.

CRB_POSITION_NOT_FOUND

No pointer to read position found.

BAD_ENT_KEY

Given key(s) is nil.

NO_MORE_ROOM

No more core memory.

MALRMV

FUNCTION: Remove an entity from a list. If KEY1 is an entity, then remove it from the constituent list of that entity.

FORMAT: MALRMV(KEY1,IPOS,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities from which an entity is to be removed.

IPOS : INTEGER

The position, in the list, of the entity to be removed.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list then an entity is removed from the list. If KEY1 is an entity then an entity is removed from the constituent list of KEY1. IPOS is the position number of the entity to be removed. The MAS delete rules are used to see if the entity can be removed from the constituent list.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_POSITION	Remove position = 0 or is greater than length.
	BAD_LIST_REFERENCE	Given key is not an entity or a list.
	INVALID_DELETE	Delete rule prohibits delete.
	INVALID_CRB_POSITION	Pointer to position of last read of constituent list is inconsistent.
	CRB_ENTRY_NOT_FOUND	Pointer to read position not found.
	RULE_DOES_NOT_MATCH	Rules incorrectly defined.
	NO_MORE_ROOM	No more core memory.

UM 560130001
1 January 1987

MALROR

FUNCTION: Reorder a list of entities so that the users appear at the head of the list.

FORMAT: MALROR(KEYL, IRC)

INPUT:

KEYL : LISTKEY
Key of an application list.

OUTPUT:

RC : INTEGER
Return code
0 = Good return
<0 Critical error
>0 Warning

DESCRIPTION: For each member of the list, search each of the remaining members for its users; put users at the head of the list.

MALRPL

FUNCTION: Replace an entity in a list. If KEY1 is an entity then replace in the constituent list of that entity.

FORMAT: MALRPL(KEY1,KEY2,IPOS,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities in which an entity is to be replaced.

KEY2 : ENTKEY

The entity that will replace an entity in KEY1.

IPOS : INTEGER

The position of the entity in KEY1 to be replaced.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then an entity is replaced in the list. If KEY1 is an entity, then an entity is replaced in the constituent list of KEY1. KEY2 must be an entity key. The entity at position IPOS in KEY1 will be replaced by KEY2. If the entity being replaced is "MARKED FOR DELETE," then an attempt is made to delete the entity.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_POSITION	Remove position = 0 or is greater than length.
	BAD_LIST_REFERENCE	Given key is not an entity or a list.
	INVALID_DELETE	Delete rule prohibits delete.
	INVALID_CRB_POSITION	Pointer to position of last read of constituent list is inconsistent.
	CRB_ENTRY_NOT_FOUND	Pointer to read position not found.
	RULE_DOES_NOT_MATCH	Rules incorrectly defined.
	NO_MORE_ROOM	No more core memory.

MALRVS

FUNCTION: Reverse the order of the entities in a list.

FORMAT: MALRVS(KEY1,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities in which the order of the entities is to be reversed.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: KEY1 may be either an entity key or a list key. If KEY1 is a list, then the list is reversed. If KEY1 is an entity, then the constituent list is reversed.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key is nil or not an entity or a list.
	NO_MORE_ROOM	No more core memory.
WARNING:	NO_LIST_CREATED	Given list is empty.

MALSRT

FUNCTION: Sorts an entity list using the order given in a user defined function.

FORMAT: MALSRT(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY

The list of entity or applications to be sorted.

KEY2 : The name of the user defined function for ordering the list

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: This routine references a user defined function which provides the order sequence to be applied to the list to be sorted.

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key is nil or not an entity or a list.
	NO_MORE_ROOM	No more core memory.
WARNING:	NO_LIST_CREATED	Given list is empty.

4.4.6 SEQUENTIAL READ AND EXECUTE OPERATIONS

The following table shows routines that process a list sequentially (as if it were a file):

DESCRIPTION	ROUTINE
Read the next entry in a list.	MALRD
Setup for reading in a forward direction.	MALSTF
Setup for reading in reverse direction.	MALSTR
Execute a procedure on an entity or a list of entities.	MAEXEQ
Execute a procedure on all entities of a specified KIND.	MAKXEQ
Execute a procedure on an entity or a list of entities.	MALXEQ
Execute a given procedure on constituents of entity.	MAECXQ
Execute a procedure on the users of an entity.	MAEUXQ

The MALSTF and MALSTR set up a list for forward or reverse reading of an application list. Forward reading is assumed and need not be called explicitly before a read or an execute function is used. However, after an end-of-list is signaled, the list is disabled. An explicit setup must be done to enable the list.

MALRD

FUNCTION: Read the next entry in a list.

FORMAT: MALRD(KEY1,KEY2,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to be read.

OUTPUT:

KEY2 : ENTKEY

The entity of the next list entry. Next depends on the direction of the read set by MALSTF or MALSTR.

IRC : INTEGER

The return code.

DESCRIPTION: The next entity in the list is returned. Always set the direction by using MALSTF or MALSTR before the first time this routine is used to read a list.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key is nil or not an entity or a list.

INVALID_CRB_POSITION

Pointer to position of last read of constituent list is inconsistent.

CRB_ENTRY_NOT_FOUND

Pointer to read position not found.

NO_MORE_ROOM

No more core memory.

MALSTF

FUNCTION: Setup for reading in forward direction.

FORMAT: MALSTF(KEY1,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to be processed in a forward direction.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: If KEY1 is an entity, then the constituent list of KEY1 will be set up.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key is not an entity or a list.

BAD_ENT_KEY

Given key is nil.

INVALID_CRB_POSITION

Pointer to position of last read of constituent list is inconsistent.

CRB_ENTRY_NOT_FOUND

Pointer to direction of read not found.

NO_MORE_ROOM

No more core memory.

MALSTR

FUNCTION: Setup for reading in reverse direction.

FORMAT: MALSTR(KEY1,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to be processed in the reverse direction.

OUTPUT:

IRC : INTEGER

The return code.

DESCRIPTION: If KEY1 is an entity then the constituent list of KEY1 will be setup.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key is not an entity or a list.

BAD_ENT_KEY

Given key is nil.

INVALID_CRB_POSITION

Pointer to position of last read of constituent list is inconsistent.

CRB_ENTRY_NOT_FOUND

Pointer to direction of read not found.

NO_MORE_ROOM

No more core memory.

The application-supplied procedure invoked by the "execute" functions must conform to the declaration shown below:

Procedure PROC(KEY,ENTDEF,DATA,IRC)

INPUT:

- 1) KEY is an entity key.
- 2) ENTDEF is the entity ADB declaration.
- 3) DATA is a variant data structure used as needed by the procedure. DATA is the input data structure passed originally to an EXECUTE function.

OUTPUT:

- 1) IRC is the return code produced by the "PROC".

The application procedure called from MAEXEQ and MAKXEQ have the following return code values:

RCC ≥ 0 and RCC ≤ 7
processing OK
The EXECUTE routine continues processing.

RCC ≥ 8 and RCC ≤ 15 .
procedure_code_error
The EXECUTE routine halts processing.

RCC < 0 or RCC > 15
procedure_out_of_range
The EXECUTE routine halts processing.

The application procedure called from MALXEQ has the following return code values:

RCC = 0 or RCC = 1
processing OK
The EXECUTE routine adds an entity to the output list and continues processing.

RCC ≥ 2 and RCC ≤ 7
procedure_warning_code
The EXECUTE routine continues processing.

RCC ≥ 8 and RCC ≤ 15
procedure_code_error
The EXECUTE routine halts processing.

MAEXEQ

FUNCTION: Execute a procedure on a entity or a list of entities.

FORMAT: MAEXEQ(KEY1,DATA,PROC,RCC,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities on which the application procedure should be performed.

DATA : VARIANT

The application defined data structure which either supplies or receives the values operated on by the application defined procedure.

PROC : ENTRY POINT

The entry point of an application defined procedure.

OUTPUT:

RCC : INTEGER

The procedure PROC return code.

IRC : INTEGER

The MAS return code.

DESCRIPTION: The entity, or each entity in a list, is passed to the application-defined procedure. The operation performed on the entity is determined by the application-defined procedure.

EXAMPLE: See Sample Programs Section.

ERRORS:

Message

Description

BAD_LIST_REFERENCE

Given key is nil or not an entity or a list.

PROC_CODE_ERROR

PROC_OUT_OF_RANGE

NO_MORE_ROOM

Return error code greater than 15.

No more core memory.

MAKXEQ

FUNCTION: Execute a procedure on all entities of a specified kind.

FORMAT: MAKXEQ(KIND,DATA,PROC,RCC,IRC)

INPUT:

KIND : INTEGER

The KIND value of the entities to be processed.

DATA : VARIANT

The application-defined data structure, which either supplies or receives the values operated on by the application-defined procedure.

PROC : ENTRY POINT

The entry point of an application-defined procedure.

OUTPUT:

RCC : INTEGER

The procedure PROC return code.

IRC : INTEGER

The MAS return code.

DESCRIPTION: Each entity of the specified kind is passed to the application-defined procedure. The operation performed on the entity is determined by the application-defined procedure.

ERRORS:

Message

Description

BAD_LIST_POSITION

Schema inconsistent.

PROC_CODE_ERROR

PROC_OUT_OF_RANGE

Returned error code greater than 15.

NO_MORE_ROOM

No more core memory.

WARNING:

NO_SUCH_SCHEMA

No definition for given kind.

MALXEQ

FUNCTION: Execute a procedure on a entity or a list of entities. Construct an output list of entities as determined by the application procedure.

FORMAT: MALXEQ(KEY1,DATA,PROC,KEY2,RCC,IRC)

INPUT:

KEY1 : ANYKEY

The entity or list of entities to be processed.

DATA : VARIANT

The application-defined data structure, which either supplies or receives the values operated on by the application-defined procedure.

PROC : ENTRY POINT

The entry point of an application-defined procedure.

OUTPUT:

KEY2 : LISTKEY

The list created by this function.

RCC : INTEGER

The procedure PROC return code.

IRC : INTEGER

The return code produced by this operation.

DESCRIPTION: An empty list (KEY2) is created. The entity, or each entity in sequence if a list is supplied, is passed to the application-defined procedure. The operation performed on the entity is determined by the application-defined procedure. When the application return code of "success," (0 or 1), is returned from the application procedure, the entity just processed is added to the result list. When an application error return code (less than 0 or greater than 7) is returned from the application procedure, MALXEQ is terminated. When an application warning return code (2 through 7) is returned from the application procedure, the entity just processed is not placed on the result list, but processing continues.

MALXEO (Cont.)

ERRORS:	<u>Message</u>	<u>Description</u>
	BAD_LIST_REFERENCE	Given key is not an entity or a list.
	BAD_ENT_KEY	Given key is nil.
	PROC_CODE_ERROR	
	PROC_OUT_OF_RANGE	Returned error code greater than 15.
	INVALID_CRB_POSITION	Pointer to position of read of the constituent list inconsistent.
	CRB_POS_NOT_FOUND	Pointer to position of read not found.
	NO_MORE_ROOM	No more core memory.
WARNING:	PROC_WARNING_CODE	
	NO_LIST_CREATED	No entities executed sucessfully.

MAECXQ

FUNCTION: Given a user-defined procedure, perform this procedure on the constituents of an entity or list of entities.

FORMAT: MAECXQ(KEY1, DATAREC, PROCNM, KEY2, RRC, IRC)

INPUT:

KEY1 : ANYKEY
Key of an entity or an application list that's constituent(s) are to be processed.

DATAREC : BLKDATA
Data to be supplied to the procedure.

PROCNM : ROUTINE
Routine supplied by caller that processes one entity at a time.

OUTPUT:

KEY2 : LISTKEY
Key to the list of constituents that processed without error.

RRC : INTEGER
Return code of the user-defined procedure.

IRC : INTEGER
Return code
0 - Good return
<0 Critical error
>0 Warning

DESCRIPTION: For each constituent of an entity read from the position and in the direction indicated in its user constituent list, process by the user-defined procedure. For each entity processed without error, add to the output list.

MAEUXQ

FUNCTION: Given a user-defined procedure, perform this procedure on the users of an entity or list of entities.

FORMAT: MAEUXQ(KEY1, DATAREC, PROCNM, KEY2, RCC, IRC)

INPUT:

KEY1 : ANYKEY
Key of an entity or an application list that's user(s) are to be processed.

DATAREC : BLKDATA
Data to be supplied to the procedure.

PROCNM : ROUTINE
Routine supplied by the caller that processes one entity at a time.

OUTPUT:

KEY2 : LISTKEY
Key to the list of users that processed without error.

RCC : INTEGER
Return code of the user-defined procedure.

IRC : INTEGER
Return code
0 = Good return
<0 Critical error
>0 Warning

DESCRIPTION: For each user of an entity or an entity on the list of entities, process by the user-defined procedure. For each user processed without error, add to the output list.

SECTION 5
GENERAL UTILITIES

This section contains descriptions of available general utility routines, as shown in the table below.

DESCRIPTION	ROUTINE
Get number of different KIND values in the working-form model.	MAECHK
Get KIND value stored at specific position in KIND list.	MAEKND
Determine if an entity has any users.	MAEUSR
Get actual model space used and amount of model free space.	MASMSZ
Determine the number of entities in the model of a specified KIND.	MAKCNT

MAECHK

FUNCTION: Get the number of different KIND values in the working-form model.

FORMAT: MAECHK(KNDCNT,IRC)

INPUT:
NONE

OUTPUT:
KNDCNT: INTEGER
The number of different KIND values in the working-form model.

IRC : INTEGER
The return code.

DESCRIPTION: Get the number of KIND values in the working-form model from the KIND list.

NOTE: Works in conjunction with MAEKND.

UM 560130001
1 January 1987

MAEKND

FUNCTION: Get KIND value at specified position in the KIND list.

FORMAT: MAEKND(KNDPOS,KNDVAL,IRC)

INPUT:

KNDPOS: INTEGER

The position in the standard array of where to get
the KIND value

OUTPUT:

KNDVAL: INTEGER

The KIND value retrieved from the KIND list

IRC : INTEGER

The return code.

DESCRIPTION: Get the KIND value at KNDPOS in the KIND list.

NOTE: Works in conjunction with MAECLK.

MAEUSR

FUNCTION: Determine if an entity has any users.

FORMAT: MAEUSR(KEY1,UEXIST,IRC)

INPUT:

KEY1 : ENTKEY

The entity whose user existence is to be determined.

OUTPUT:

UEXIST: INTEGER

The value indicating if the entity has users or not.

-0 No users exist

-1 Users exist

IRC : INTEGER

The return code.

DESCRIPTION: Determines if an entity has any users.

MASMSZ

FUNCTION: Determine actual model used space and model free space (in bytes).

FORMAT: MASMSZ(MODSIZ,FRESIZ,IRC)

OUTPUT:

MODSIZ : INTEGER
The total number of bytes used by the model.

FRESIZ : INTEGER
The total number of bytes of free space.

IRC : INTEGER
The return code.

DESCRIPTION: The used model space is calculated by taking the difference of allocated model space and free model space. This routine can only be used where the MAS memory manager is used.

MAKCNT

FUNCTION: Determine the number of entities in the model of a specified KIND.

FORMAT: MAKCNT(KIND,COUNT,IRC)

INPUT:

KIND : INTEGER
The KIND value for which a count is to be determined.

OUTPUT:

COUNT : INTEGER
The number of entities in the model of the specified KIND.

IRC : INTEGER
The return code.

DESCRIPTION: If the KIND specified is in the model, determine the number of entities with that KIND.

SECTION 6
SAMPLE PROGRAMS

The following pages illustrate uses of the Access Software. These examples show Create and Get operations for a line.

DESCRIPTION	MAS ROUTINES USED
Create a Line	MAL, MALATC, MAECR, MALD
Get Constituents	MALNO, MALGTK
Get Users	MAEU, MALNO, MALGTK

Line Get Users

```

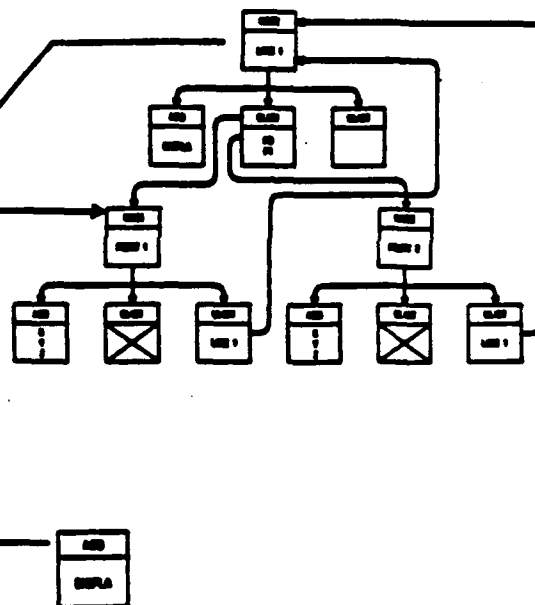
PROCEDURE USERS :
VAR
  BUFFER      : ENTBLOCK ;
  INC         : EXT_RET_CODE ;
  KEYE_SMS    : ENTKEY ;
  KEYLC       : LISTKEY ;
  NEXT_USER   : INTEGER ;
  NUM_OF_USERS : INTEGER ;
BEGIN
  . SELECT ENTITY
  (* GET USER LIST *)
  MAEU (KEYE_SMS,KEYLC,INC) ;
  (* GET NUMBER IN LIST *)
  MALND (KEYLC,NUM_OF_USERS,INC) ;

  WHILE NUM_OF_USERS > 0 DO BEGIN
    (* GET ENTITY FROM LIST*)
    MALGX (KEYLC,NEXT_USER, KEYE_SMS,INC) ;

    (* GET ADD FROM ENTITY*)
    MAESTX (KEYE_SMS,BUFFER,INC) ;

    . DISPLAY ENTITY INFORMATION
    .
    NEXT_USER := NEXT_USER + 1 ;
    NUM_OF_USERS := NUM_OF_USERS - 1 ;
  END ;
END ;

```



Line Get Constituents

EXAMP.DAT(CNSTNTS)

PROCEDURE CNSTNTS;

VAR

 BUFFER : ENTBLOCK ;
 INC : EXT_RET_CODE ;
 CNSTNT_KEY : ENTKEY ;
 KEYLC : LISTKEY ;
 NEXT_CNSTNT_NO : INTEGER ;
 NUM_OF_CONSTITUENTS : INTEGER ;
 NUM_OF_DATA_ENT : INTEGER ;
 RESPONSE : SSRESPONSE ;

BEGIN

 VERIFY INPUT AND INITIALIZE

 NEXT_CNSTNT_NO := 1 ;

 (* DETERMINE NUMBER OF CONSTITUENTS *)

 MALNO (INPUTKEY, NUM_OF_CONSTITUENTS, INC) ;

 WHILE NUM_OF_CONSTITUENTS > 0

 DO BEGIN

 (* GET ENTITY KEY FROM LIST *)

 MALSTX (INPUTKEY, NEXT_CNSTNT_NO,
 CNSTNT_KEY, INC) ;

 (* GET ENTITY ADD *)

 MAESTX (CNSTNT_KEY, BUFFER, INC) ;

 USE THE DATA IN THE ADD

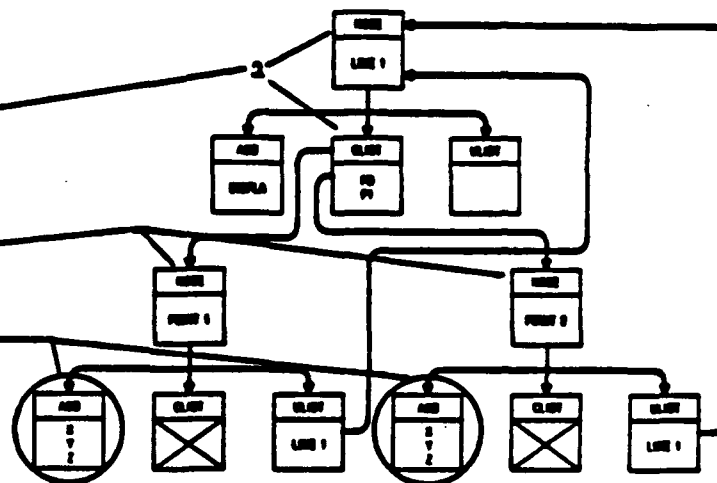
 NEXT_CNSTNT_NO := NEXT_CNSTNT_NO + 1 ;

 NUM_OF_CONSTITUENTS :=

 NUM_OF_CONSTITUENTS - 1 ;

 END ;

END ;



SPRINGER

Line Create an Entity

EXAMP.DATA(CNETLM)

CC

KIND = KLINE
LENGTH = LLINE
SYSUSE = 0
VERSUM = 0
SYSD = IDENT
IDENT = IDENT
DISPLA = DISPLY
RGB(1) = 0
RGB(2) = 0
RGB(3) = 0
INTENS = 0
SYMBOL = DISTYP

C

C

C

CREATE AN EMPTY LIST

CALL MAL (LIST,IRC)

IF(IRC .NE. 0) GOTO 900

C

C

C

CREATE CONSTITUENT LIST OF LINE

DO 100 I = 1,MLINE

CALL MALATC(LIST,KEYPNT(I),IRC)

100 CONTINUE

C

C

C

CREATE A LINE ENTITY

CALL MAECR(ADB, LIST, KEY, IRC)

C

IF(IRC .NE. 0) GOTO 900

C

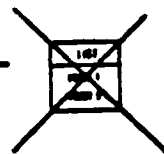
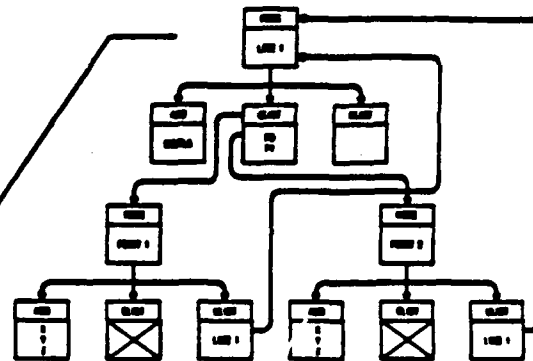
C

C

DELETE LIST

CALL MALD(LIST,IRC)

C



0000-0000-0000

APPENDICES

INTRODUCTION

Valuable supplementary information not included earlier in this manual is included in this section.

	<u>Page</u>
A ACCESS SOFTWARE CALLING PARAMETER INDEX.	A-1
B ALPHABETICAL ACCESS SOFTWARE ROUTINE INDEX	B-1
C ACCESS SOFTWARE RETURN CODE INDEX.	C-1
D ACCESS SOFTWARE FORTRAN SCHEMA DIAGRAM	D-1
E ACCESS SOFTWARE PASCAL SCHEMA DIAGRAM.	E-1
F GENERAL TECHNIQUES/GUIDELINES.	F-1
G RUN-TIME ENVIRONMENT	G-1
INTRODUCTION.	G-1
INTERLANGUAGE CONVENTIONS	G-2
ESTABLISHING INTERLANGUAGE ENVIRONMENT.	G-3
REGISTER CONVENTIONS.	G-5
PASCAL DYNAMIC STORAGE AREA	G-6
EXAMPLES.	G-8
H ERROR AND WARNING RETURN CODE INDEX.	H-1

ACCESS SOFTWARE CALLING PARAMETER INDEX

<u>Routine</u>	<u>Description and calling sequence</u>	<u>Page</u>
MAINIT	Initialize the working-form model MAINIT (ext_ret_code)	4-5
MAKILL	Delete the current working-form model MAKILL (ext_ret_code)	4-6
MAECR	Create an entity MAECR (entblock, anykey, entkey, ext_ret_code)	4-9
MAEC	Create list of constituents MAEC (anykey, listkey, ext_ret_code)	4-10
MAECI	Create list of inclusive constituents MAECI (anykey, listkey, ext_ret_code)	4-11
MAECIK	Create list of inclusive constituents by kind MAECIK (anykey, ord_kind, listkey, ext_ret_code)	4-12
MAEU	Create list of users MAEU (anykey, listkey, ext_ret_code)	4-13
MAEUI	Create list of users inclusively MAEUI (anykey, listkey, ext_ret_code)	4-14
MAEUIK	Create list of users inclusively by kind MAEUIK (anykey, ord_kind, listkey, ext_ret_code)	4-15
MAEGKN	Get kind value of an entity MAEGKN (entkey, integer, ext_ret_code)	4-17
MAEGTK	Get entity ADB MAEGTK (entkey, entblock, ext_ret_code)	4-18
MAEUD	Update the ADB MAEUD (entkey, entdef, ext_ret_code)	4-20
MAED	Delete an entity or list of entities MAED (anykey, listkey, ext_ret_code)	4-24
MAEDI	Delete an entity or list of entities inclusively MAEDI (anykey, listkey, ext_ret_code)	4-25
MAEDT	Delete test an entity or list of entities MAEDT (anykey, listkey, listkey, ext_ret_code)	4-26
MAEDTI	Delete test an entity or list of entities inclusively MAEDTI (anykey, listkey, listkey, ext_ret_code)	4-27
MAEDTS	Delete test an entity or list of entities (output - 3 lists) MAEDTS (anykey, listkey, listkey, listkey, ext_ret_code)	4-28
MAEA	Activate an entity or list of entities MAEA (anykey, ext_ret_code)	4-30
MAEAI	Activate an entity or list of entities inclusively MAEAI (anykey, ext_ret_code)	4-31
MAEAV	Find value of entity activation setting MAEAV (entkey, integer, ext_ret_code)	4-32

ACCESS SOFTWARE CALLING PARAMETER INDEX

<u>Routine</u>	<u>Description and calling sequence</u>	<u>Page</u>
MAERST	Set application flag in all entities in model to "off" MAERST (namtyp, ext_ret_code)	4-34
MAQURY	Determine value of application flag of the entity MAQURY (entkey, namtyp, integer, ext_ret_code)	4-35
MAUPDT	Update value of application flag of entity or list of entities MAUPDT (anykey, namtyp, integer, ext_ret_code)	4-36
MAECQY	Determine whether the user compresses with its constituent MAECQY (entkey, entkey, integer, ext_ret_code)	4-37
MAECMP	Create a list of constituents which compress with the input entity MAECMP (entkey, listkey, ext_ret_code)	4-38
MAESWA	Set all entities binary switch setting to "off" MAESWA (ext_ret_code)	4-39
MAESWT	Set binary switch in an entity or list of entities MAESWT (anykey, integer, ext_ret_code)	4-40
MAESVL	Find binary switch setting of an entity MAESVL (entkey, integer, ext_ret_code)	4-41
MALAND	"And" of two lists MALAND (anykey, anykey, listkey, ext_ret_code)	4-44
MALNOT	"Not" of two lists MALNOT (anykey, anykey, listkey, ext_ret_code)	4-45
MALOR	"Or" of two lists MALOR (anykey, anykey, listkey, ext_ret_code)	4-46
MALK	Create list of an entities of specified kind MALK (ord_kind, listkey, ext_ret_code)	4-48
MALKL	Create list of an entities of specified kind which are found within another list MALKL (anykey, ord_kind, listkey, ext_ret_code)	4-49
MAL	Create an empty list MAL (listkey, ext_ret_code)	4-51
MALN	Create an empty list of specified size MALN (integer, listkey, ext_ret_code)	4-52
MALCPY	Make a copy of a list MALCPY (listkey, listkey, ext_ret_code)	4-53
MALFND	Find position of an entity in a list MALFND (anykey, entkey, integer, integer, ext_ret_code)	4-54
MALNO	Count entities in a list MALNO (anykey, integer, ext_ret_code)	4-55
MALGTK	Get the Nth entity from a list MALGTK (anykey, integer, entkey, ext_ret_code)	4-56

ACCESS SOFTWARE CALLING PARAMETER INDEX

<u>Routine</u>	<u>Description and calling sequence</u>	<u>Page</u>
MALD	Delete a list MALD (listkey, ext_ret_code)	4-58
MALDA	Delete all lists in working-form model MALDA (ext_ret_code)	4-59
MALDI	Delete alist and all lists after it MALDI (anykey, ext_ret_code)	4-60
MALOCK	Set the list lock flag MALOCK (listkey, integer, ext_ret_code)	4-61
MALATC	Attach entity or list of entities to entity or list MALATC (anykey, anykey, ext_ret_code)	4-63
MALINS	Insert entity or list of entities into a list MALINS (anykey, anykey, integer, ext_ret_code)	4-64
MALRDE	Remove duplicate entities from list MALRDE (listkey, ext_ret_code)	4-65
MALREP	Replace a list of entities MALREP (anykey, anykey, ext_ret_code)	4-66
MALRMV	Remove entity or list of entities MALRMV (anykey, integer, ext_ret_code)	4-67
MALROR	Reorder a list of entities so that the users appear at the head of the list MALROR (listkey, ext_ret_code)	4-68
MALRPL	Replace an entity MALRPL (anykey, entkey, integer, ext_ret_code)	4-69
MALRVS	Reverse order of list MALRVS (anykey, ext_ret_code)	4-70
MALSRT	Sorts an entity list using a user defined function MALSRT (anykey, routine, ext_ret_code)	4-71
MALRD	Read next entity in list MALRD (anykey, entkey, ext_ret_code)	4-73
MALSTF	Set flag to read in forward direction MALSTF (anykey, ext_ret_code)	4-74
MALSTR	Set flag to read in reverse direction MALSTR (anykey, ext_ret_code)	4-75
MAEXEQ	Execute procedure on an entity or list of entities MAEXEQ (anykey, blkdata, routine, ext_ret_code, ext_ret_code)	4-77
MAKXEQ	Execute procedure on all entities of specified kind MAKXEQ (anykey, blkdata, routine, ext_ret_code, ext_ret_code)	4-78
MALXEQ	Execute procedure on entity or list of entities MALXEQ (anykey, blkdata, routine, listkey, ext_ret_code, ext_ret_code)	4-79
MAECXQ	Perform this procedure on the constituents of an entity or list of entities, given a user defined function MAECXQ (4-81
MAEUXQ	Perform this procedure on the users of an entity or list of entities, given a user defined function	4-82

ACCESS SOFTWARE CALLING PARAMETER INDEX

<u>Routine</u>	<u>Description and calling sequence</u>	<u>Page</u>
MAECHK	Get number of different kinds in working-form model MAECHK (integer, ext_ret_code)	5-2
MAEKND	Get kind value at specified position in kind list MAEKND (integer, ord_kind, ext_ret_code)	5-3
MAEUSR	Determine if an entity has any users MAEUSR (entkey, integer, ext_ret_code)	5-4
MASMSZ	Find actual model used space and model free space MASMSZ (integer, integer, ext_ret_code)	5-5
MAKCNT	Determine number of entities in model of specified kind MAKCNT (integer, integer, ext_ret_code)	5-6

ALPHABETICAL ACCESS SOFTWARE ROUTINE INDEX

Routine	Description	Page
MAEA	Activate an entity or list of entities	4-30
MAEAI	Activate an entity or list of entities inclusively	4-31
MAEAV	Find value of entity activation setting	4-32
MAEC	Create list of constituents	4-10
MAECI	Create list of inclusive constituents	4-11
MAECIK	Create list of inclusive constituents by kind	4-12
MAECR	Create an entity	4-9
MAECMP	Create a list of constituents with which the input entity compresses	4-38
MAECTK	Get number of different kinds in working-form model	5-2
MAECQY	Determine whether the user compresses with its constituent	4-37
MAECXQ	Given a user-defined procedure, perform this procedure on the constituents of an entity or list of entities	4-81
MAED	Delete an entity or list of entities	4-24
MAEDI	Delete an entity or list of entities inclusively	4-25
MAEDT	Delete test an entity or list of entities	4-26
MAEDTI	Delete test an entity or list of entities inclusively	4-27
MAEDTS	Delete test an entity or list of entities (output - 3 lists)	4-28
MAEGKN	Get kind value of an entity	4-17
MAEGTK	Get entity ADB	4-18
MAEKND	Get kind value at specified position in kind list	5-3
MAERST	Set application flag in all entities in model to "off"	4-34
MAESVL	Find binary switch setting of an entity	4-41
MAESWA	Set all entities binary switch setting to "off"	4-39
MAESWT	Set binary switch in an entity or list of entities	4-40
MAEU	Create list of users	4-13
MAEUD	Update entity ADB	4-20
MAEUI	Create list of users inclusively	4-14
MAEUIK	Create list of users inclusively by kind	4-15
MAEUSR	Determine if an entity has any users	5-4
MAEUXQ	Given a user-defined procedure, perform this procedure on the users of an entity or list of entities	4-82
MAEXEQ	Execute procedure on an entity or list of entities	4-77
MAINIT	Initialize the working-form model	4-5
MAKCNT	Determine number of entities in model with specified kind	5-6
MAKILL	Delete the current working-form model	4-6
MAKXEQ	Execute procedure on all entities of specified kind	4-78
MAL	Create an empty list	4-51
MALAND	"And" of two list	4-44
MALATC	Attach entity or list of entities to entity or list	4-63
MALCPY	Make a copy of a list	4-53
MALD	Delete a list	4-58
MALDA	Delete all lists in the working-form model	4-59
MALDI	Delete a list and all lists after it	4-60

ALPHABETICAL ACCESS SOFTWARE ROUTINE INDEX

<u>Routine</u>	<u>Description</u>	<u>Page</u>
MALFND	Find position of an entity in a list	4-54
MALGTK	Get the Nth entity from a list	4-56
MALINS	Insert entity or list of entities into a list	4-64
MALK	Create list of an entities of specified kind	4-48
MALKL	Create list of an entities of specified kind which are found within another list	4-49
MALN	Create an empty list of specified size	4-52
MALNO	Count entities in a list	4-55
MALNOT	"Not" of two lists	4-45
MALOCK	Set the list lock flag	4-61
MALOR	"Or" of two lists	4-46
MALRD	Read next entry in list	4-73
MALRDE	Remove duplicate entities from list	4-65
MALREP	Replace list of entities	4-66
MALRMV	Remove entity or list of entities	4-67
MALROR	Reorder a list of entities so that the users appear at the head of the list	4-68
MALRPL	Replace entity or list of entities	4-69
MALRVS	Reverse the order of a list	4-70
MALSRT	Sorts an entity list using a user defined function	4-71
MALSTF	Set flag to read in forward direction	4-74
MALSTR	Set flag to read in reverse direction	4-75
MALXEQ	Execute procedure on entity or list of entities	4-79
MAQURY	Determine value of application flag for given entity	4-35
MASMSZ	Find actual model used space and model free space	5-5
MAUPDT	Update value of application flag of entity or list of entities	4-36

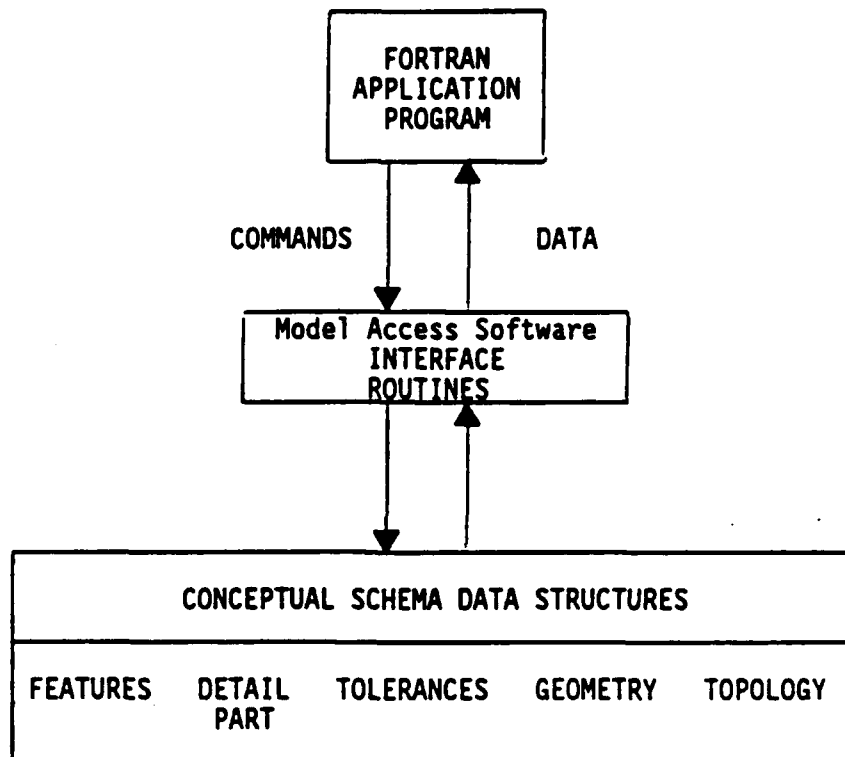
ACCESS SOFTWARE RETURN CODE INDEX

<u>Error type</u>	<u>Code</u>
NO_ERRORS_DETECTED	0
BAD_ENT_KIND	1
INVALID_CREATE	2
CANT_CREATE_LIST	3
MAS_INIT_FAILED	4
INVALID_UPDATE	5
CANT_UPDATE_ENT	6
CANT_CREATE_ENT	7
CANT_VERIFY_CONNECT	8
INVALID_CONNECTION	9
CANT_CONNECT	10
ABSENT_INPUT	11
INVALID_GET	12
NDS_OP_COMPLETE	13
BAD_LIST_POSITION	14
MAXIMUM_LIST_SIZE	15
BAD_LIST_MOVE_COUNT	16
BAD_LIST_REFERENCE	17
BAD_ENT_KEY	18
DUPLICATE_SCH	19
DUMP_ERROR	20
BAD_ENT_SIZE	21
BAD_SCH_KIND	22
PROC_CODE_ERROR	23
PROC_OUT_OF_RANGE	24
NO_MATCH_FOUND	25
DUPS_NOT_REMOVED	26
INVALID_DELETE	27
BAD_ENTITY_ON_USER_LIST	28
BAD_DELETE_KEY	29
EMPTY_MODEL	30
ARG_OUT_OF_RANGE	31
INVALID_CRB_POSITION	32
CRB_ENTRY_NOT_FOUND	33
INVALID_FLAG_NAME	34
CANT_MARK_ENTITY_DELETE	35
SIZE_NOT_CARE_ENOUGH	36
RTS_NOT_IN_WORKING_FORM	37
CORE_NOT_AVAILABLE	38
NOT_ENOUGH_CORE_FOR_INIT	39
ABSOLUTELY_NO_MORE_CORE	40

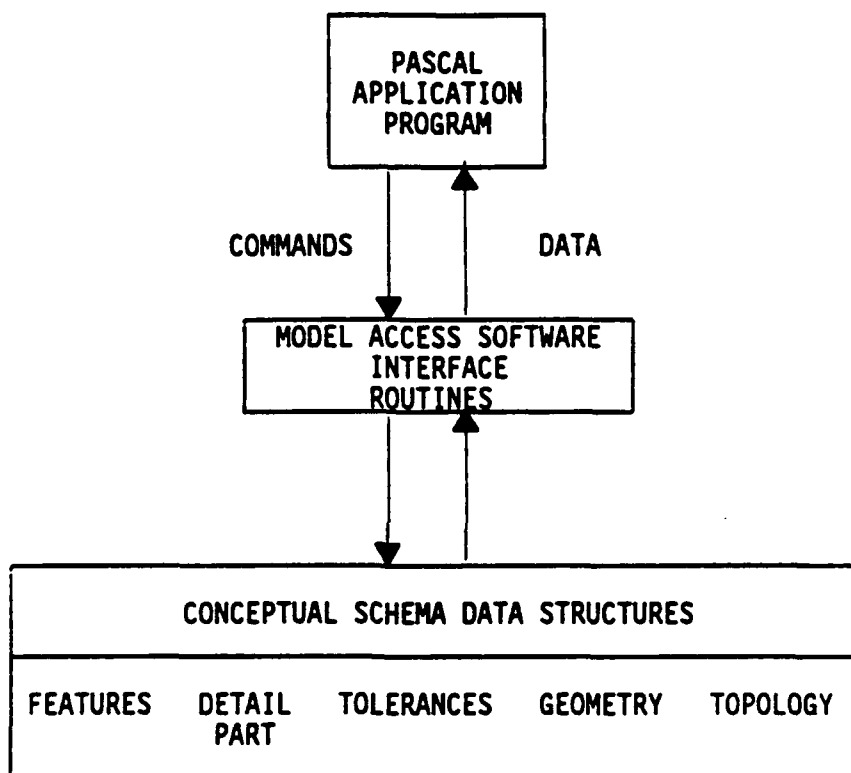
ACCESS SOFTWARE RETURN CODE INDEX

<u>Warning type</u>	<u>Code</u>
OKW	0
NO_SUCH_SCH	-1
PROC_WARNING_CODE	-2
EMPTY_DELETE_LIST	-3
EMPTY_EXCEPTION_LIST	-4
END_OF_LIST	-5
NO_LIST_CREATED	-6
EMPTY_MARK_LIST	-7

ACCESS SOFTWARE FORTRAN SCHEMA DIAGRAM



ACCESS SOFTWARE PASCAL SCHEMA DIAGRAM



GENERAL TECHNIQUES/GUIDELINES

- o Avoid creating long lists of entities:
 - Lists are processed sequentially
 - Lists use model space
- o Do not use ENTKEY as a memory address:
 - ENTKEY does not address the attribute data block of the entity
- o Avoid "nil" keys:
 - Abend or nil pointer checking errors may be caused
- o Delete application lists when no longer needed:
 - Application lists use memory
 - Application lists slow deletion of entities
- o Always test the MAS interface return code:
 - RC = 0 normal return
 - RC < 0 warning message
 - RC > 0 critical error
- o Reset the process bit to "off" when it is no longer needed.
- o Define the KIND and LENGTH fields in the ADB.
- o When MALRD is used in conjunction with one of the following interface routines:

MAED
MAEDI
MAL

MALINS
MALRMV

the position of sequential reading is incremented/decremented if an interface function modifies the list.

Do not use MALGTK and one of the above routines because the local variable position cannot be adjusted by the MAS package.

For example:

```
VAR NUM_IN_LIST: = INTEGER  
BEGIN  
  FOR I = 1 TO NUM_IN_LIST    DO  
    MALGTK (LISTKEY, NUM_IN_LIST, ENTKEY1):  
    MAED (ENTKEY1, LISTX):  
END:
```

As each entity is deleted, it is removed from the LISTKEY list, but I is not adjusted.

- o With the exception of MAL and MALK, empty lists will not be created. If an interface function has an output LISTKEY and the list is empty, the list will not be created and the LISTKEY will be NIL. A warning return code will indicate this situation.

RUN-TIME ENVIRONMENT

INTRODUCTION

The Access Software consists of a set of Pascal procedures that provides an interface to the working form model for application programs. When the application programs are written in a language other than Pascal, the run-time environment must satisfy the interlanguage communication requirements of all the languages involved. This appendix discusses the MAS interlanguage environment conventions and the composition of the Pascal dynamic storage areas. Examples are given for a FORTRAN program that uses MAS routines.

INTERLANGUAGE CONVENTIONS

When the MAS subprograms were compiled, they were defined as PROCEDURES using SUBPROGRAM declarations. The subprogram declaration is an extension to IBM Pascal that allows a Pascal procedure to be called from any language. The subprogram declaration supplies special code at compile time. At run-time, this code determines the nature of the calling program. For non-Pascal calls, two macros are invoked: Prolog and Epilog. Before the procedure executes, Prolog locates the Pascal Communication Work Area (PCWA) as well as the main and local Dynamic Storage Areas (DSA) and establishes the Pascal register conventions. On exit, the Epilog macro restores the register conventions of the calling program.

The effect of this method is that no special action is required by the calling program, regardless of its language.

The SUBPROGRAM declaration may also be applied to application procedures, which may then be called from, and make calls to, routines of any language. This method is limited to Pascal PROCEDURES and does not apply to Pascal FUNCTIONS.

ESTABLISHING INTERLANGUAGE ENVIRONMENT

The preferred (and easiest) approach is to insert the entire application into a Pascal program. This method assures correct error handling.

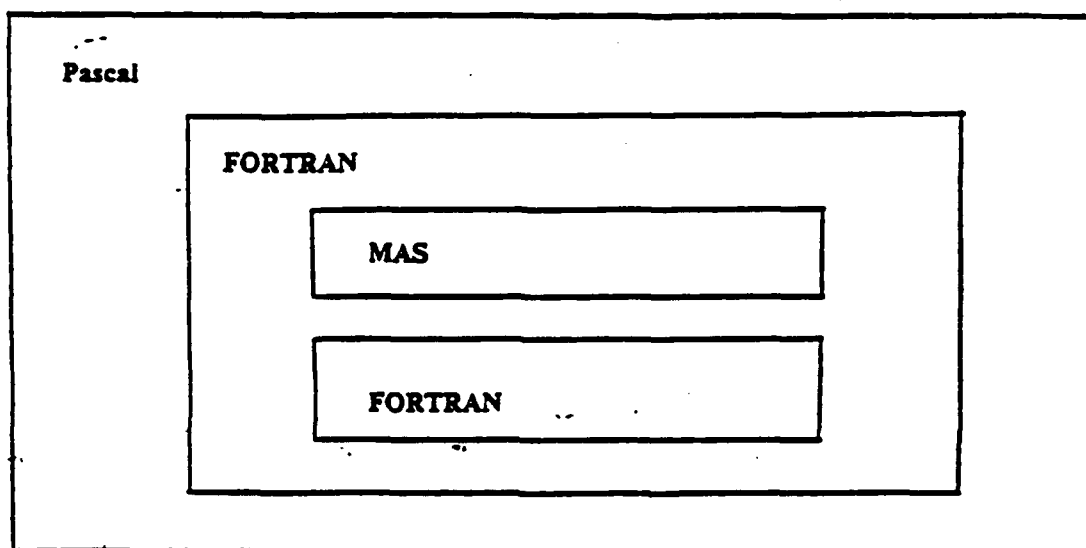


Figure G-1

An alternate approach is to insert the portion of the application that makes the MAS calls into a Pascal procedure that is declared MAIN. The error handling capability, however, may be limited in this method. Note that the model created within the scope of the MAIN Pascal procedure is active only during the execution of the MAIN procedure; new models may be created in subsequent calls to a similarly declared MAIN procedure. Upon termination of the last call to a Pascal MAIN, the procedure PSCLHX should be called to terminate the Pascal run-time environment.

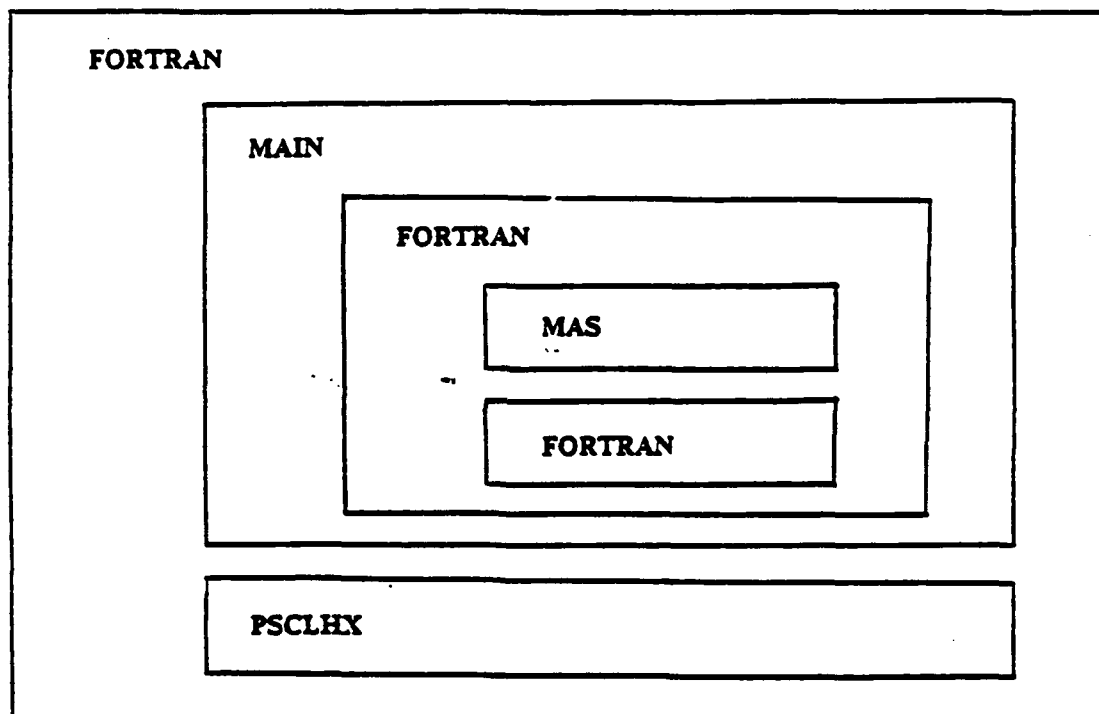


Figure G-2

Examples of the Pascal source and link-edit instructions are included at the end of this appendix.

REGISTER CONVENTIONS

The interlanguage environment establishes the correct register conventions automatically. The following information is included for use from the IBM TEST mode.

<u>Register</u>	<u>Pascal</u>	<u>Non-Pascal</u>
15	Branch address	Branch address
14	Return address	Return address
13	Local DSA address (1)	Save area address
12	PCWA address	
11	Main DSA address	
1	Address of parameter list (2) (3)	Address of parameter list
0	(2)	Function value

- NOTES:
- (1) The save area is the first entry in the local DSA, which is established by a Pascal caller.
 - (2) The function value for Pascal is referred to by the first entry in the parameter list. Pascal input parameters for a function are referred to as starting with the second entry in the parameter list.
 - (3) The parameter list contains addresses of parameters except for pass-by-value of scalars, pointers, or sets, in which case the parameter list contains the actual value.

PASCAL DYNAMIC STORAGE AREA

The dynamic storage area of the Pascal main program contains global variables (including any commons). Each Pascal procedure invoked has a local dynamic storage area containing local variables. The dynamic storage areas are contained in a LIFO stack.

In general, the DSA of a routine consists of five sections:

- (1) The local save area (144).
- (2) Parameters passed in by the caller.
- (3) Local variables required by the routine.
- (4) A save area required by any routine that will be called.
- (5) Storage for the largest parameter list to be built for a call.

Sections 1 and 2 are allocated by the calling routine; Sections 3, 4, and 5 are allocated by the Prolog of the called routine.

Every DSA is at least 144 bytes long. This is the storage required by Pascal/VS for a save area. The local variables and parameters of the routine are mapped within the DSA starting at offset 144.

Upon entering a routine, Register 1 points 144 bytes into the DSA of the routine, which is where the parameters passed in by the caller reside.

Upon invocation, Register 13 points to the base of the DSA of the caller, which is where the save area of the caller is located. Figure 3 illustrates the condition of the stack and relevant registers immediately upon the start of the routine.

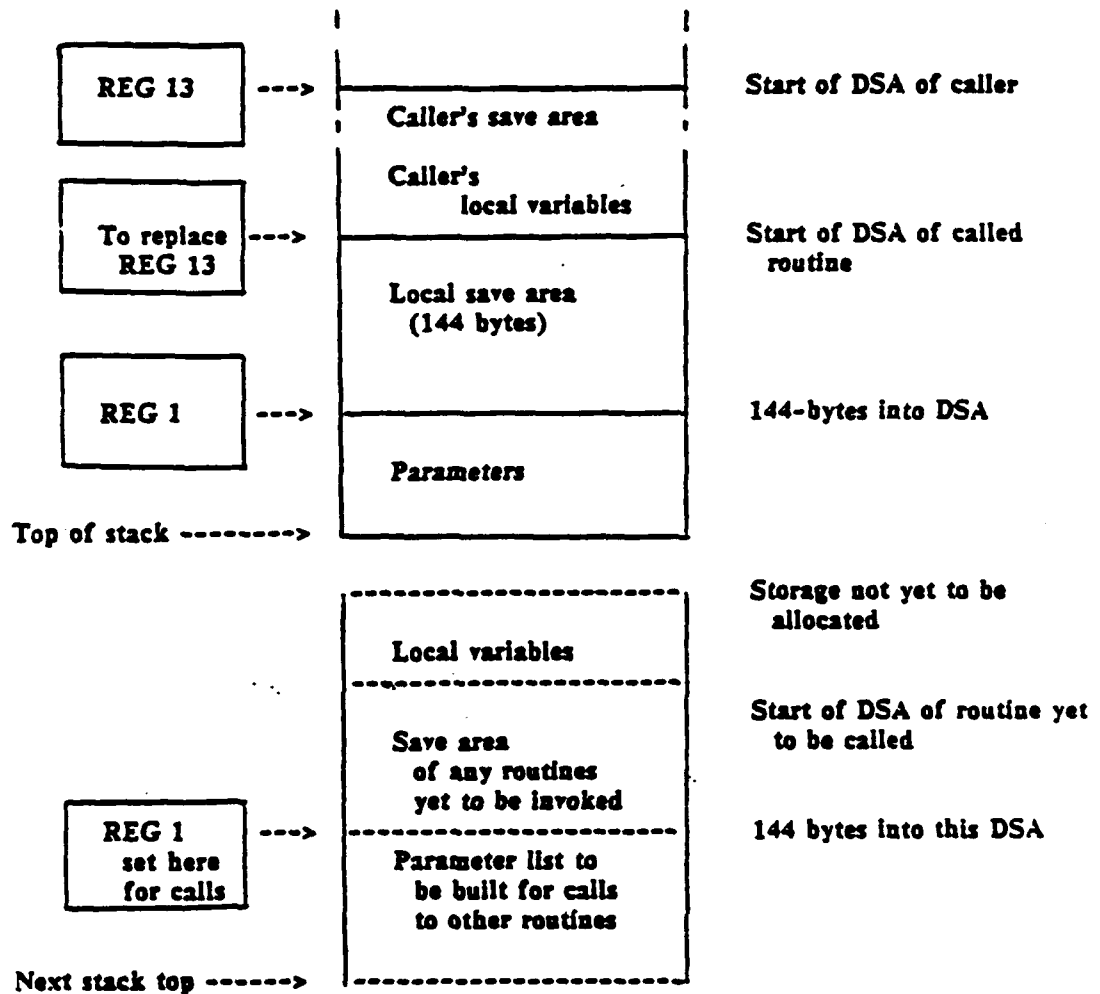


Figure G-3

EXAMPLES

EXAMPLE 1: PASCAL PROGRAM (PASMAIN) THAT INVOKES FORTRAN MAIN

```
PASCAL PROGRAM
PROGRAM PASMAIN;
PROCEDURE MAIN; FORTRAN;
BEGIN
MAIN;                Invoke FORTRAN main.
END.
```

```
LINKEDIT INSTRUCTIONS
INCLUDE APLLIB(PASMAIN)
INCLUDE APLLIB(APL    )    FORTRAN main object,
                             list of objects including FORTRAN main,
                             or LOAD module including FORTRAN main.

INCLUDE MASLIB(MAS    )
ENTRY  PASMAIN
```

where SYSLIB allocation includes SYS1.PASCLIB.

EXAMPLE 2: PASCAL PROCEDURE (PASSUB) INVOKED BY FORTRAN MAIN THAT INVOKES
FORTRAN SUBROUTINE (FORSUB)

PASCAL PROCEDURE

SEGMENT PASSUB;

PROCEDURE PASSUB (....);MAIN;

FORTRAN MAIN may pass parameters to the
PASCAL subroutine.

PROCEDURE PASSUB;

PROCEDURE FORSUB(....);FORTRAN;

PASCAL MAIN may pass parameters to the
FORTRAN MAIN.

BEGIN

FORSUB(....);

Invokes FORTRAN subroutine that calls
MAS.

end;

FORTRAN MAIN PROGRAM

.
.
.
CALL PASSUB(....)
CALL PSCLHX
.
.
.

LINKEDIT INSTRUCTIONS

INCLUDE APPLIB(APL)

List of objects including FORTRAN MAIN
or LOAD module including FORTRAN MAIN.

INCJDE APPLIB(PASSUB)

INCLUDE MASLIB(MAS)

ENTRY APL

NAME APL

where SYSLIB allocation includes SYS1.PASCLIB.

MAS ERROR AND WARNING RETURN CODE INDEX

ERROR TYPE	CODE	ERROR TYPE	CODE
NO_ERRORS_DETECTED	0	ABSOLUTELY_NO_MORE_CORE	40
BAD_ENT_KIND	1	MAINIT_ALREADY_DONE	41
INVALID_CREATE	2	RULE_DOES_NOT_MATCH	42
CANT_CREATE_LIST	3	ENTITY_NOT_FOUND_LIST	43
MAS_INIT_FAILED	4		
INVALID_UPDATE	5		
CANT_UPDATE_ENT	6		
CANT_CREATE_ENT	7		
CANT_VERIFY_CONNECT	8		
INVALID_CONNECTION	9		
CANT_CONNECT	10		
ABSENT_INPUT	11		
INVALID_GET	12		
NDS_OP_COMPLETE	13		
BAD_LIST_POSITION	14		
MAXIMUM_LIST_SIZE	15		
BAD_LIST_MOVE_COUNT	16		
BAD_LIST_REFERENCE	17		
BAD_ENT_KEY	18		
DUPLICATE_SCH	19		
DUMP_ERROR	20		
BAD_ENT_SIZE	21		
BAD_SCH_KIND	22		
PROC_CODE_ERROR	23		
PROC_OUT_OF_RANGE	24		
NO_MATCH_FOUND	25		
DUPS_NOT_REMOVED	26		
INVALID_DELETE	27		
BAD_ENTITY_ON_USER_LIST	28		
BAD_DELETE_KEY	29		
EMPTY_MODEL	30		
ARG_OUT_OF_RANGE	31		
INVALID_CRB_POSITION	32		
CRB_ENTRY_NOT_FOUND	33		
INVALID_FLAG_NAME	34		
CANT_MARK_ENTITY_DELETE	35		
SIZE_NOT_LARGE_ENOUGH	36		
RTS_NOT_IN_WORKING_FORM	37		
CORE_NOT_AVAILABLE	38		
NOT_ENOUGH_CORE_FOR_INIT	39		